Microbee HACKER'S HANDBOOK

$6.50*
N.Z. $6.95

PEEK

POKE

Hard and soft projects for Bees of all vintages

Everything a Hackers micro should bee

- Make speech
- Make music
- Communicate
- Experiment
- And enjoy
- with microbee

microbee®

CONTENTS

Cover Photograph: Courtesy of Tom Moffat (hacker extraordinaire).

Editor Roger Harrison
Production Editor Natalie Filatoff
Art and Design Brian Jones
Ali White
Editor-in-chief Jane Mackenzie
Advertising
Richard Pakalnis
(Group Advertising Manager)
David McDowall
(National Advertising Manager)
Managing Editor Jim Rowe
Publisher Michael Hannan

Offices
NSW
140 Joynton Avenue, Waterloo 2017
(02) 663-9999
Telex: FEDPUB AA74488

VICTORIA
150 Lonsdale St., Melbourne 3000
(03) 662-1222
Telex: FEDPUB AA34340

WESTERN AUSTRALIA
Ian Greenacre, Gordon & Gotch Adv't,
134-144 Parry St., Perth 6000
(09) 328-8044

QUEENSLAND
Craig Santilla, Media Services,
4 Adelaide Terrace, Brisbane 4000
(07) 229-6033

SOUTH AUSTRALIA
Steve Birbeck, Admedia Group,
24 Kensington Rd., Rose Park 5067
(08) 332-8144

NEW ZEALAND
Chris Horsley, 4A Symonds Court,
Symonds St., Auckland NZ
Telex: TEXTURE 260753

Distributed nationally by Gordon and Gotch.

*** Recommended and maximum price only.**

ISBN 0 86405 082 8

So! You've done it again! You've gone out and bought yet another computer book. Just wait till you get home – "There you go, spending more money on that §†%¢ ‡%'§* computer while the rest of us can't even afford a night at the movies any more."*

Yes my friend, like it or not, you are a confirmed hacker. In the early days of computerism, to be called a 'hacker' was an insult: it implied an unkempt slob, a bore with never a thought except for the holy machine. Today, the meaning of the term has flipped 180 degrees.

Now, the hacker is held in considerable awe among more conventional computer people. The hacker is the person who digs deep below the surface of BASIC or Pascal or whatever, into the bowels of the computer's operating system. Hackers may not be completely expert with the computer's high-level language because they seldom use it. To hackers, a high-level language is really a useful collection of machine code sub-routines.

This book celebrates the art of hacking as applied to the Microbee. It's almost as if the Microbee was designed specifically with hackers in mind. Like Doolittle's famed Pushmi-pullyu, its rear end is just as interesting as its front. At the back you will find up to five sockets, carrying a total of nearly 100 pins. Every one of them is there to allow you to put information into, or take information from, your Microbee. Many sections of this book will show you how to 'hook things up'.

The Microbee comes from a company called Applied Technology, whose main claim to fame used to be its series of circuit board kits, which could be plugged together to form a complete computer – sort of. Hackers loved them. They bought the kits and made 'improvements' as they assembled them. There were some weird computer systems around back in those days.

Applied Technology also turned out kits for the 'Dream' computer, a learner's micro. Many hackers developed their addictions on these machines, and even today, amidst all the IBMs and Apples, Dreams are cherished possessions. I certainly won't be parting with mine.

The Microbee was dropped upon us in a supplement to 'Your Computer' magazine in February, 1982. There, in glowing colour, was a picture of a new computer kit with a full keyboard instead of the Dream's 16 hexadecimal keys – and it ran BASIC.

I wanted one, everyone wanted one, but for a long time nobody got anything. Applied Technology was caught with its proverbial pants down; it just couldn't deliver the landslide of orders. Every phone call was met with the response Jerry Pournelle always relates in 'Byte': "Real soon now ... " But, to make a long story short, the Bees eventually started flowing, new

FOREWORD

models came out, and Applied Technology grew and grew.

The kit Microbees didn't stay kits for long. They were quite fiddly to build, and a sleeping disaster for the first-time kit builder. Soon, ready-made became the rule, but not before lots of hackers enjoyed the satisfaction of building and showing off a home-made computer.

The first model change came with the 'Plus'. This was simply a normal Microbee with a parallel port plug and memory battery fitted as standard, instead of being optional. Further model changes were mostly internal in nature, involving such things as using fewer memory chips, each with bigger capacity; in other words, modernising.

Then came the IC model (Integrated Computer). This one had a new version of BASIC that supported colour commands, and, most importantly for the hacker, a change in clock speed. The earlier Bees used a 2 MHz clock; the IC changed it to 3.375 MHz. Suddenly all existing software ran faster, and software that depended on certain time delays, such as my RTTY and FAX programs, became all wrong (swear, grumble, grumble). But it looks like the change was made with a purpose in mind ... to write characters to the screen faster, so 80 of them could be laid down during normal television horizontal scan times.

With the new 80-character by 24-row screen, the Microbee joined the big league. Networking, telecomputing, disk drives, CP/M software – all became available to the Bee user. The very latest 128K Microbee comes with dual disk drives, an optional amber monitor in matching decor, bundled software, and so on. But the hackers haven't been forgotten. The main computer still has those familiar 100 pins on the back.

Tom Moffat

Computers in Schools

AT LAST! A PUBLICATION FOR TEACHERS, PARENTS, STUDENTS . . . AND ANYONE CONCERNED WITH THIS VERY IMPORTANT ISSUE.

ISSUES:

Does your child really need a home computer?
It's become rather difficult to see through all the media hype! We provide down-to-earth answers to major questions (Will your child by handicapped without a computer? Or even handicapped by one?)

The push into the schools market
An impartial look at the wave of advertising power currently breaking over the (supposedly) lucrative schools market. Who really stands to gain — the kids, or computer manufacturers?

VIEWPOINTS:

What the experts say
Recognised authorities in the fields of both education and computing give their opinions.

How the teachers feel
Computers are likely to feature strongly in the future of education. We tell how teaching staff are coming to grips with this fact.

What the kids think
Are computers worthwhile? Just good fun? Or a waste of time? Students comment (candidly) on how they feel about computer education.

PRACTICALITIES:

How to pick the right system
For high schools, primary schools.

Educational software
What's available; what's really educational; what to look for when shopping.

Networking and schools
What it is; how it works; how it can help.

Computer aided education in practice
Getting right down to basics: how are computers actually used in the schoolroom?

Computers in administration. The out-of-the-classroom aspect. How computers can streamline the behind-scenes running of a school.

CASE HISTORIES:

The UK scene
How far has computer education advanced overseas? And how far will it go?

True-life stories
From schools around Australia — how computers are working for them.

All this, plus a useful directory of services, and a simple-language glossary to help you in your purchase of a computer.

AVAILABLE JANUARY, 1985

Letter From Applied Technology

It is a pleasure to see publications such as this becoming available for the Microbee.

The Microbee was introduced in the February 1982 edition of 'Your Computer' and has become the most popular Australian personal computer to date. Today there are over 30,000 Microbees in daily use in Australia and worldwide, and the demand is growing daily. There are 39 Microbee user groups active in Australia, with others in Sweden, New Zealand, Israel and several other countries around the world. The Microbee has become a user's machine and a lot of these people will gladly accept the tag 'hacker', for it was from the ranks of 'hackers' that the design of the Microbee arose.

It is the owners who collectively determine whether a computer is a success or not, and their enthusiasm depends largely on the degree of support available to them. The continual support of magazines such as 'Electronics Today' and 'Your Computer' has contributed significantly to the unique success of the Microbee. Another factor is the continual user interaction with the product, and this is evidenced by the scope of the contents of this handbook.

Tom Moffatt's articles on the RTTY Decoder and the FAX Decoder deserve special mention in my opinion because they represent world firsts. Where else can you find informative articles on building what is virtually the receiving section of a home facsimile machine operating from radio broadcasts?

Users worldwide will welcome this 'Hacker's Handbook' as it represents a useful collection of articles in one bound volume, which will become a reference for years to come. At Applied Technology we are delighted to co-operate with the publishers of this much needed publication.

Owen Hill
Managing Director,
Applied Technology.

Playing With the Microbee's

Have you ever wondered how to make your Microbee display 80 characters across the screen, or change its cursor, or produce more rows of text on the screen? Eric Lindsay explains how these changes can be accomplished from within your programs.

UNFORTUNATELY IT IS difficult to explain how the 6545 video chip in the Microbee works, but we can see a few of its effects with a simple POKE to memory. Since many other computers such as the IBM-PC, the Big Board II and some Apple video boards use similar chips, you may be able to use similar techniques with them.

If you experiment a lot, you may end up with your video display completely out of control. Remember you can always fix everything up by doing a 'cold' start (press <ESC> and <RESET> for a few seconds, so that you get the 'Micro-World' copyright message on your screen). Now to work!

POKE 220,97 and you suddenly have a full block-style cursor. POKE 2200,111 gets it back to normal. Values between 97 and 111 will give different heights for the cursor. Normally things are not this easy to change, and in most of the cases I describe you will have to load values into the 6545 chip. Luckily you can use the existing routines in the Microbee to do this, just by putting the values you want to change into memory, and then doing a 'warm' start (this just means you press the <RESET> key for a second or two, and release it to get a 'READY' message).

When you do a cold start, your Microbee moves certain values from its BASIC EPROMs into its user memory and loads these values into the 6545 video display controller. If you look in the BASIC EPROMs at location 47820 (BACC hex), and at the next 15 memory locations, you will find the initial values loaded into registers R0 to R15 of the 6545. These values are moved to user memory at 210 to 225 (D2 hex to E1 hex), where we can change them easily. All this is rather dry and not very exciting, so let's try to make a few changes.

Moving Your Display

PRINT PEEK (212) produces the value 75. POKE 212,65 or POKE 212,85, and then do a warm start by pressing <RESET>. This will move your video display starting point sideways, the same as is done by <ESC> <A> or <ESC> <S>. You can check this by using <ESC> <A> or <ESC> <S>, and then PRINT PEEK (212), and comparing the results. You can move your entire video display up and down by POKE 217,17 or POKE 217,19. This is the location used by <ESC> <W> and <ESC> <Z>. Again, to see the result, you have to press <RESET>. POKE 217,18 to restore the correct value.

Video Display

The cursor can be altered via locations 220 and 221, and you have already seen some of the changes that are possible. Memory location 220 contains the number of the scan line at which the cursor will start, while 221 contains the number of the scan line at which it will end. It normally starts and ends at scan line 15, thus giving a single-line cursor at the bottom of each line. However, you can put any numbers between 0 and 15 into these locations, and have your cursor of any height and at any position in the line of characters. Just try POKE 220,5:POKE 221,10 and you will see what I mean.

Back already? No doubt you are wondering why the cursor is no longer blinking? Well, I cheated a little, because memory location 220 originally contained 111, not the 15 I told you. However, 111 is 64 + 32 +15. Ask a silly question . . . If you add 64 to your POKE (POKE 220,64+15) you get a blinking cursor. If you add only 32 you get no cursor at all, and if you add 64 and 32 you get a blinking cursor, but at different speed. All easy, provided you remember to press <RESET> after poking the memory.

Now we come to the hard bit: changing the number of characters in a line, and the number of lines on the screen. It's hard because you have to understand, in outline, how a television or monitor works. Otherwise you will just have a list of POKEs, and no idea of how to change them to suit your own needs.

How A Television Works

When a microcomputer presents anything on a video display, it has to send a signal to the display telling it to make a dot of light wherever a dot should appear. Monitors or televisions make up their display from many separate pictures, or frames. A television makes 25 frames each second. Each frame is made up of about 625 horizontal lines. You usually don't see this many, because some of them are lost in the black band which appears when a television picture rolls up and down the screen.

Since having only 25 frames tends to produce a flickering effect, much like old-fashioned movies, each frame is split into two sets of 312 lines which are interlaced, so that each alternate frame shows the odd or the even lines. This reduces the flicker to an acceptable level. In microcomputer displays, the odd and even frames are mostly identical, although it is possible to vary this, as you will see later.

Now, to display anything on the screen, the microcomputer has to turn a dot on and off along each scan line. It has to produce enough dots to fill the entire line (512 for the Microbee), and then turn off the dots at the end of the line. It then has to signal the television to go to the next scan line, leave enough time for the television to do so, and start producing the dots for the next line, and so on to the bottom of the display. There it has to turn off the lines, tell the television to start the next frame, and leave enough time for it to do so. All these exercises have to be precisely timed to suit the television or monitor, and of course all the signals produced have fancy names.

We can call the scan line the Horizon- ▶

Details of the Microbee's 6545 CRT Registers.

REG No.	REGISTER Name	R/W?	ROM Hex	CONTENTS Hex	CONTENTS Dec	RAM Hex	RAM Hex	Description of register
00	Horiz Total	W	BAC0	5F	095	D2	210	Horizontal total, to take 64000 nS
01	Chars/Row	W	BAC1	40	064	D3	211	No of displayed characters
02	HYSNC Pos	W	BAC2	4B	075	D4	212	Horz Screen Position ESC A S
03	H/V SYNCH Wd	W	BAC3	37	3+7	D5	213	Hsync/Vsync for monitor
04	Vert Total	W	BAC4	12	018	D6	214	No of Rows, to equal 313 scan lines
05	VSYNC Adjust	W	BAC5	09	009	D7	215	for adjustment of scan line no
06	CharRows/Frame	W	BAC6	10	016	D8	216	Character rows to display
07	VSYNC Pos	W	BAC7	12	018	D9	217	Vsynch, ESC W Z value
08	Mode Control	W	BAC8	48	072	DA	218	Allows interlaced scan, etc.
09	ScanLines/Row	W	BAC9	0F	015	DB	219	Adj this for 24 lines, etc.
10	Cursor Start	W	BACA	6F	111	DC	220	Cursor blink & start
11	Cursor End	W	BACB	0F	015	DD	221	Scan line where cursor ends
12	Start Address	W	BACC	00	0	DE	222	MSB Top of screen pace
13	Start Address	W	BACD	00	0	DF	223	LSB "
14	Cursor Pos	R/W	BACE	00	0	E0	224	MSB Cursor Position
15	Cursor Pos	R/W	BACF	00	0	E1	225	LSB "
16	Light Pen Pos	R	-	-	-	-	-	MSB Light Pen
17	Light Pen Pos	R	-	-	-	-	-	LSB position
18	Update Loc	W	-	-	-	-	-	Update screen address
19	Update Loc	W	-	-	-	-	-	to be accessed next

Playing With the Bee's Video Display

tal Total, and it includes the time to return to the next line (the Retrace time). The signal the computer sends to end this is called the Horizontal Sync Pulse, and this can be of varied duration, called the Horizontal Width. The length of time the whole frame takes is the Vertical and Total, since this tends not to be set too accurately, it is 'fine tuned' by a Vertical Sync Adjustment. The resulting figure is the time for the whole frame, including the time in which nothing is displayed.

We can also adjust the number of rows of characters displayed, and when the frame ends (the Vertical Sync Signal). As with the Horizontal Sync, there is also a Vertical Sync Width adjustment, used to make it easier to suit signals to a particular television or monitor. When you move the display left or right on the screen you are really adjusting the Horizontal Sync timing, and when you move it up or down the screen, you are adjusting the Vertical Sync timing.

Now we have to add a few calculations. We know we produce 50 frames a second, of 312 lines. We also know that each character in the Microbee is 16 lines high (count them), and that we produce 16 rows of characters. This means we see 256 of the 312 lines. We also know that we can produce 64 characters, and that each can be up to eight dots wide, or 512 visible dots on each line.

I will tell you now (would I lie?) that we need enough time on each line to produce 96 characters, allowing for retrace times and other delays. We also know, from the circuit diagrams supplied by Applied Technology, that the 6545 chip has a 12 MHz signal, which is divided by eight, giving 1.5 MHz (in the original description in the February 1982 *Your Computer* this was said to be 1.25 MHz, but if that is correct, then all my calculations stop working).

The result of all this is that you give a signal to the 6545 to produce a new character every 666.7 nanoseconds, thus taking 64,000 nanoseconds per scan line of 96 characters, or 20,000,000 (1/50 second) per frame, or one second for 50 frames. Well, all the mathematics work out pretty well. Now we have to look at how the 6545 chip uses this sort of thing.

Inside The 6545

In the 6545, register 0 contains the number of characters per line, less one. PRINT PEEK (210) will give 95, implying that 96 character times are used per scan line. Because we need to keep this timing pretty much correct, this figure should not be altered.

Register 1, at memory location 211, normally contains the number of characters displayed, or 64. This can be changed to any number you please. However, if you change it by very much, you may also have to adjust the position of the horizontal sync pulse, which is set by register 2 at location 212. You will also find that although you can POKE 211,80, you will not see 80 characters because some of them will run off the end of the display.

Since the BASIC believes the display is producing 64 characters, you will find that each line starts in a different place. To correct this, the actual BASIC EPROMs must be changed.

Register 3 contains the horizontal and vertical sync widths, is set to 55, and should not be altered. These settings are used to match the signals from the computer to those needed by monitors; in this case it produces a horizontal sync pulse that takes the same time as three characters, and a vertical sync pulse that takes the same time as seven scan lines.

Things are more complicated when we consider the vertical timing. Register 4, set by location 214, contains the number of rows of characters per screen. It is set to 18 because it includes rows we don't see. Also, the figure it contains is one less than the number of rows produced, so it actually produces 19 rows of characters.

I mentioned that each character is 16 scan lines high. This figure minus 1 is placed in register 9, at location 219, so this contains 15. Between the two of them, these registers must produce 313 scan lines, otherwise the television or monitor will roll badly. A little light calculation music, and we multiply 16 scan lines by 19 rows and get 304.

Luckily we still have register 5, the vertical sync adjustment register at location 215, which increases the number of scan lines by whatever number it contains. It contains 9, and thus everything works out. You can change the contents of all three of these registers, if the number of rows (contents of register 4, plus 1) multiplied by the number of scan lines per character (contents of register 9, plus 1), when added to the contents of register 5, equals 313.

Now, we turn to register 6, at location 216. This contains 16, which is the number of rows of characters we want to display. You can reduce this freely and get fewer rows on the screen. However, if you want to increase it, you have problems. For one thing, BASIC knows it can display only 16 rows; and for another, we still only have a limited number of lines on the screen.

Let's change that. POKE 214,24: POKE 215,1: POKE 216,24 or 23: POKE 217,23 or 24 or 25 – you may have to experiment – POKE 219,12. Now press <RESET>. With a bit of luck, you will have a stable display with 16 lines all squashed up into the top and middle of the screen, and random shapes on the bottom third of the display. BASIC is still only showing 16 lines of text, since it puts text only in the first 1000 bytes of your video memory. The bottom of the screen is displaying the contents of the second 1000 bytes of your video memory. You can POKE your own messages into this area fairly easily, as you can see by starting with something like FOR X=63488 to 64000:POKEX,32:NEXT X.

Incidentally, one of the things we just did was to ensure that each character displayed was only 12 scan lines high. This means you have lost the tails (descenders) on characters like j and y. It also means that if you really demand 24 lines on the display, you have to change the character generator supplied with the Microbee (luckily this is fairly easy, but still not a task for beginners).

While there are a host of other interesting registers in the 6545, I will deal with only one more of them, and that is register 8, at location 218. This is the mode control register, and normally contains 72. Try changing this to 73 or 74 or 75. Changing to 73 provides an interlaced scan. This means that each character is only half as high as before, and you could theoretically fit 48 lines on the screen with ease. Unfortunately, the display usually shakes so badly in this mode that it makes viewing intolerable.

Details of all the registers on the 6545 chip are included in the chart on the previous page. Experiment a bit, even though you'll find some of the changes you can make will have no effect apart from stopping you video display from working. Remember, you cannot hurt your Microbee by playing with the keyboard, and you can fix your video by doing a cold start. ■

In response to Eric Lindsay's article (see previous pages), which was originally published in 'Your Computer' February 1984, we received the following instructions from C. Angus and D. Edmonds on running the Microbee screen in the 80-character by 25-line mode, but still under full control of BASIC.

A 25-Line by 80-Character Screen in BASIC

YES, your Microbee can run a professional screen in BASIC, despite what everyone has said to the contrary! If you studied Eric Lindsay's article 'Playing with Your Microbee's Video Display' you will already know the way to reconfigure the Microbee screen, but not how to interface it to BASIC.

Reproduced at the end of this article is a machine code program which reconfigures the 6545 CRT controller chip and provides a new VDU driver routine which is protected from BASIC, such that a NEW, LOAD or a WARM START will not destroy it. But first a word of warning ...

Back-up Foul-up

We wouldn't be surprised if many people playing with the registers of the 6545 become convinced their machine has developed a real fault, despite assurances that any mistakes may be obliterated by a cold start. Unfortunately, the 'light pen' registers of the 6545 are used to scan the keyboard. Normally, this can only be regarded as a clever piece of engineering on the part of Applied Technology; but, if an invalid combination of values is written into the 6545 it no longer scans the keyboard, so when you press Reset and Escape to do a cold start, your Microbee does not 'see' the ESC key and so only does a warm start! Turning the power off and on again does not help, as most Microbees are fitted with battery back-up. We spent several hours over one machine, with logic analysers and so on, until the great truth was realised ... you have to *disconnect the battery*! Our machines are now fitted with battery switches, which saves a great deal of hair-tearing.

Explaining the Program

Now, to the program. You have probably noted the six small sections before the actual VDU routine. Let's deal with them in order of appearance. This program is loaded into the top of memory, so it is protected from BASIC by making the machine think it has less memory than actually fitted. BASIC uses a 'top of memory' pointer in location 00A0/A1, so if we reserve the top 200 (hex) bytes for playing around, we can load this pointer with 3DFF (hex) – that is for 16K Microbees, of course. Owners of 32K machines would use 7DFF (hex) and code the program from 7E00. All internal jumps are 'relative', so no recompiling should be necessary.

Next, the necessary values are loaded into the allocated scratch area by lines 260 to 290, and line 340 calls a subroutine in BASIC ROM to load these values into the 6545 registers. Lines 380 to 420 change an address in the BASIC scratch area (address 000B2/B3), which is the 'jump vector' normally pointing to the VDU driver routine in the BASIC ROMs. The start address of the new driver routine is inserted here.

As was pointed out in Eric's article, we reduce the number of TV lines to twelve per character line, so normally the descenders of lower-case characters are cropped. To overcome this, lines 460 to 490 copy the PCG (Programmable Character Generator) RAM into itself, but shifted four addresses lower. This effectively raises all the characters into full view. It is assumed that the PCG is in the INVERSE mode, either from a cold start or by invoking it under BASIC. This does mean your new display will be entirely in inverse video; if you don't like this, you could re-write the lines to re-invert the data.

The last mini-section which returns control to BASIC may seem a bit strange, since a value is pushed onto the stack, but this is just a carbon copy of how this entry point is reached under BASIC and it is desirable to keep the pushes and pops balanced. This is a ▶

A 25 x 80 Screen

warm start point, so the program can be called before running a BASIC program without destroying it.

At last, we come to the actual screen-displaying routine. Entry is at line 600 and the ASCII code for the character to be displayed is passed in the 'A' register. All registers are preserved on the stack and the character code is further copied into the 'B' register, before the cursor address is transferred into the HL register pair and checked to make sure it is within the bounds of the screen. If it isn't, the clear-screen subroutine is called, which also 'homes' the cursor. It should be noted that the cursor address, stored in location 01BH, is the true RAM address starting at F000H. This could prove useful in other programs where you may want to POKE directly to the screen.

From line 740 to line 920, the ASCII code is checked in case it is a relevant screen control code. In line 770, if the code is equal to or greater than a space code (20H), the character is printable and control jumps to line 1220. The first three lines here copy the character code back into register A, but with bit seven set to a one. This means the programmable character generator will be used to write the letter to the screen, not the ROM. (We intend to copy this ROM, but shifted up, as described above, obviating this necessity and enabling mixed text and graphics under this routine.)

Line 1250 is where the ASCII code is written into screen memory; that is, it's displayed at last! After this it's more housekeeping, starting with checking that the cursor is still on the screen. If it is going off the end, the screen is scrolled up one line, and the cursor moved back 80 characters (50H) and checked to make sure it is still in the screen area. The various screen control functions are comparatively simple, with the exception of Carriage Return. This is made complex by the fact that 80 characters is not a convenient sub-multiple of 256, unlike 64 (which may well explain the Microbee's normal display).

Two useful facts may be observed about the cursor address: first, the most significant digit is always F(hex); and second, as each line is 50(hex) in length, a carriage return will always set the least significant bit to 0. This means only the middle two digits need to be processed.

As the current cursor position is held in the HL register pair, we can Rotate Right the HL pair four bits and copy L into A, ready for treatment. This is effected by lines 1660 to 1730. A is also preserved in B, and then we keep subtracting 05 from A until it is less than 05. As we got A from shifting HL, 05 represents 50(hex) and this subtraction leaves A holding a figure representing how far along the current line the cursor is. If this is subtracted from B, which holds the original value of A, the result will represent the cursor address of the start of the current line, giving the answer we need. That takes us up to line 1000. A check is done to make sure we haven't got a value which takes the cursor to the bottom of the screen, and then we do the opposite type of shuffling to get the correct value back into HL. First, H is loaded with OF(hex), and L with A; then HL is rotated left four times, giving the most significant digit a value of F(Hex); the next two digits holding the value are transferred from A, and the least significant bit is equal to 0.

So far, this is the most economic way we have found to effect this subroutine, but if anyone has a better way please let us know.

The last couple of points which need a mention are lines 2040 to 2190, which contain the address locations used in the program, as well as the ASCII control codes used, and lines 2220 to 2370 which hold the values to be loaded into the 6545 registers.

Using the Routine

That is the complete routine, but a few notes about using it are called for. First, this display absolutely fills the active picture area, but most monitors have the display expanded too much to see all the characters and so need to have the horizontal and vertical size reduced. Unfortunately, if your monitor has only a vertical adjustment, you will need to reduce the overall picture size.

You will also discover that a Clear Screen instruction (CLS) still only clears the top half of the display. This is because it is not executed by the VDU routine. However, the screen is cleared and cursor homed by a Form Feed code (0C hex or 12 dec), which may be done in the immediate mode by Control L (pressing CTRL and L keys together). Available 'control' functions are listed below.

MICROBEE 80*25 SCREEN IN BASIC ANGUS and EDMONDS

Cursor Up	-"-	O	ØF "	15 "
Curs Left	-"-	H	Ø8 "	8 "
Curs Right	-"-	N	ØE "	14 "
Curs Down	-"-	J	ØA "	1Ø "
Bell (beep)	-"-	G	Ø7 "	7 "

If you find you cannot adjust the horizontal hold of your monitor to stop the top line distorting, try altering the number of displayed lines to 24. This requires changing the values in lines 2280 and 2290 to 18 (hex). We have not provided the display repositioning facility (ESC W, ESC Z and so on), since the display already occupies all the available screen area. This does mean this routine may be short enough to patch into Tape BASIC, as used on the earlier 64K Microbees, but we haven't looked into this.

Naturally, as the PCG is being used to provide the text display, neither graphics mode should be used – as they would overwrite the PCG RAM. Another point is that the CURS instructions will put the cursor in places you may not expect, so they should be used with care.

```
ADDR   CODE     LINE   LABEL  MNEM    OPERAND
               00010          ; A routine to rearrange the Microbee VDU
               00015          ; to 25 lines of 80 characters.
               00020          ; usable from BASIC.
               00025          ; By Don Edmonds and Chris Angus
               00030          ;  10,Adisham St.
               00035          ;  Maddington,
               00040          ;  W.A. 6109.
               00045
               00050          ;  March 1984
               00055
               00060
               00100                  ;VDU DRIVER   80 * 25
               00110                  ;  by DECA
               00120                  ; 8 - 3 - 84
               00130
0400           00140          DEFR    16
3E00           00150          ORG     3E00
               00160
               00170
               00180                  ;LOWER TOP-OF-MEMORY POINTER
               00190
3E00 21FF3D    00200          LD      HL,3DFFH
3E03 22A000    00210          LD      (MEMTOP),HL
               00220
               00230
               00240                  ;LOAD 6545 VALUES INTO SCRATCH
               00250
3E06 21123F    00260 BEGIN    LD      HL,L6545
3E09 11D200    00270          LD      DE,00D2
3E0C 011000    00280          LD      BC,0010
3E0F EDB0      00290          LDIR
               00300
               00310                  ;SET DATA INTO 6545
               00320
3E11 21E100    00330          LD      HL,00E1
3E14 CDA785    00340          CALL    85A7
               00350
               00360                  ;CHANGE VDU DRIVER VECTOR
               00370
3E17 11323E    00380          LD      DE,VDUCON
3E1A 21B200    00390          LD      HL,00B2
3E1D 73        00400          LD      (HL),E
3E1E 23        00410          INC     HL
3E1F 72        00420          LD      (HL),D
               00430
               00440                  ;COPY AND MODIFY CH.ROM TO PCG RAM
               00450
3E20 2104F8    00460          LD      HL,0F804
3E23 1100F8    00470          LD      DE,0F800
3E26 010008    00480          LD      BC,0800
3E29 EDB0      00490          LDIR
               00500
               00510
               00520                  ;RETURN TO BASIC
               00530
3E2B 21E683    00540          LD      HL,83E6
3E2E E5        00550          PUSH    HL
3E2F C35F84    00560          JP      845FH
               00570
               00580                  ;VDU HANDLING PROGRAM
               00590
3E32 00        00600 VDUCON   NOP
3E33 F5        00610          PUSH    AF
3E34 C5        00620          PUSH    BC
3E35 D5        00630          PUSH    DE
3E36 E5        00640          PUSH    HL
3E37 47        00650          LD      B,A
3E38 2A0B01    00660          LD      HL,(CURSAD) ;is cursor off screen?
3E3B 7C        00670          LD      A,H
3E3C FEF0      00680          CP      0F0
3E3E 383F      00690          JR      C,CLS
3E40 CD723E    00700          CALL    SCREND  ; if yes, clear screen
3E43 303A      00710          JR      NC,CLS  ; and home cursor
               00720
               00730
3E45 78        00740 SCRINT   LD      A,B
3E46 FE7F      00750          CP      DEL     ;test for DELETE
3E48 2846      00760          JR      Z,DDL0
3E4A FE20      00770          CP      SPACE   ;test for SPACE
3E4C 3044      00780          JR      NC,DSP
3E4E FE0A      00790          CP      LF      ;test for LINE FEED
3E50 2868      00800          JR      Z,DLF
3E52 FE08      00810          CP      BS      ;test for BACK-SPACE
3E54 286C      00820          JR      Z,DBS
3E56 FE07      00830          CP      BL      ;test for BELL
3E58 2870      00840          JR      Z,DBL0
3E5A FE0C      00850          CP      FF      ;test for FORM FEED
```

A 25 x 80 Screen

```
ADDR   CODE    LINE   LABEL   MNEM    OPERAND
3E5C 2821      00860          JR      Z,CLS
3E5E FE0F      00870          CP      SO       ;test for SHIFT OUT
3E60 2835      00880          JR      Z,DSO
3E62 FE0D      00890          CP      CR       ;test for CARRIAGE RETURN
3E64 285A      00900          JR      Z,DCR0
3E66 FE0F      00910          CP      SI       ;test for SHIFT IN
3E68 2847      00920          JR      Z,DSI
               00930
               00940
3E6A CD8EA7    00950 CURS1    CALL    CURS2    ;call s/r in BASIC ROM
3E6D E1        00960          POP     HL       ;to update curs postn
3E6E D1        00970          POP     DE
3E6F C1        00980          POP     BC
3E70 F1        00990          POP     AF
3E71 C9        01000          RET              ; EXIT from VDU routine
               01010
3E72 E5        01020 SCREND   PUSH    HL       ;test for end of screen
3E73 7C        01030          LD      A,H      ; F7D0 is last screen addr.
3E74 FEF7      01040          CP      0F7
3E76 2802      01050          JR      Z,SCND
3E78 E1        01060          POP     HL
3E79 C9        01070          RET
3E7A 7D        01080 SCND     LD      A,L
3E7B FED0      01090          CP      0D0
3E7D E1        01100          POP     HL
3E7E C9        01110          RET
               01120
3E7F 2100F0    01130 CLS      LD      HL,SCRST         ;clear screen s/r
3E82 E5        01140          PUSH    HL
3E83 1101F0    01150          LD      DE,0F001
3E86 01CF07    01160          LD      BC,07CFH
3E89 3620      01170          LD      (HL),SPACE
3E8B EDB0      01180          LDIR
3E8D E1        01190          POP     HL
3E8E 18DA      01200          JR      CURS1
               01205
3E90 183A      01210 DDL0     JR      DDL      ;to keep jumps relative
               01215
3E92 AF        01220 DSP      XOR     A
3E93 CBFF      01230          SET     7,A      ;set PCG bit
3E95 A8        01240          XOR     B        ;ex or char code into A
3E96 77        01250          LD      (HL),A   ; print char to screen
3E97 23        01260 DSO      INC     HL       ; inc curs count
3E98 7C        01270 LF1      LD      A,H
3E99 CD723E    01280          CALL    SCREND   ;check for end of screen
3E9C 20CC      01290          JR      NZ,CURS1  ; if OK, get out
3E9E E5        01300          PUSH    HL       ; else, move all text up a line
3E9F 2150F0    01310          LD      HL,0F050         ;START OF 2ND LINE
3EA2 1100F0    01320          LD      DE,SCRST
3EA5 019007    01330          LD      BC,0790          ;SCREENFULL - ONE LINE
3EA8 EDB0      01340          LDIR
3EAA 2190F7    01350          LD      HL,0F790         ;START OF LAST LINE
3EAD CD093F    01360          CALL    CLRLIN   ;clear bottom line
3EB0 E1        01370          POP     HL
3EB1 11B0FF    01380 DSI      LD      DE,0FFB0         ;Shift In entry
3EB4 19        01390          ADD     HL,DE
3EB5 3EEF      01400          LD      A,0EFH
3EB7 BC        01410          CP      H
3EB8 20B0      01420          JR      NZ,CURS1
3EBA 115000    01430 DLF      LD      DE,0050          ;Line Feed entry
3EBD 19        01440          ADD     HL,DE
3EBE 18D8      01450          JR      LF1
               01455
3EC0 181F      01460 DCR0     JR      DCR      ;just to keep jumps relative
               01465
3EC2 2B        01470 DBS      DEC     HL       ;Back Space entry
3EC3 CB64      01480          BIT     4,H
3EC5 20A3      01490          JR      NZ,CURS1
3EC7 23        01500 CRSINC   INC     HL
3EC8 18A0      01510          JR      CURS1
               01515
3ECA 1809      01520 DBL0     JR      DBL      ;just to keep jumps relative
               01525
3ECC 2B        01530 DDL      DEC     HL       ;Delete entry
3ECD CB64      01540          BIT     4,H
3ECF 28F6      01550          JR      Z,CRSINC
3ED1 3620      01560          LD      (HL),SPACE
3ED3 1895      01570          JR      CURS1
               01580
3ED5 E5        01590 DBL      PUSH    HL       ;play a beep for Bell
3ED6 21A000    01600          LD      HL,00A0
3ED9 0630      01610          LD      B,30     ; call "play" s/r
3EDB CD5FA7    01620          CALL    TONGEN ; in BASIC ROM
3EDE E1        01630          POP     HL
3EDF 1889      01640 CURS10   JR      CURS1
               01650
3EE1 E5        01660 DCR      PUSH    HL       ;Carriage Return entry
```

```
ADDR   CODE    LINE   LABEL   MNEM    OPERAND
3EE2 0604      01670          LD      B,04     ;calculates next line start
3EE4 AF        01680 SHIFT1   XOR     A        ;and puts cursor there
3EE5 CB1C      01690          RR      H
3EE7 CB1D      01700          RR      L
3EE9 10F9      01710          DJNZ    SHIFT1
               01720
3EEB 7D        01730          LD      A,L
3EEC 47        01740          LD      B,A
3EED D605      01750 SUB1     SUB     05
3EEF FE05      01760          CP      05
3EF1 30FA      01770          JR      NC,SUB1
               01780
3EF3 4F        01790          LD      C,A
3EF4 78        01800          LD      A,B
3EF5 91        01810          SUB     C
3EF6 FE7D      01820          CP      7DH
3EF8 E1        01830          POP     HL
3EF9 309D      01840          JR      NC,LF1
3EFB 260F      01850          LD      H,0FH
3EFD 6F        01860          LD      L,A
               01870
3EFE 0604      01880          LD      B,04
3F00 AF        01890 SHIFT2   XOR     A
3F01 CB15      01900          RL      L
3F03 CB14      01910          RL      H
3F05 10F9      01920          DJNZ    SHIFT2
3F07 18D6      01930          JR      CURS10
               01940
               01950
3F09 0650      01960 CLRLIN   LD      B,50     ;clears a text line
3F0B 3E20      01970 L79C1    LD      A,SPACE
3F0D 77        01980          LD      (HL),A
3F0E 23        01990          INC     HL
3F0F 10FA      02000          DJNZ    L79C1
3F11 C9        02010          RET
               02020
               02030
010B           02040 CURSAD   EQU     010BH    ;holds cursor address
F000           02050 SCRST    EQU     0F000    ;start of screen ram
010D           02060 ESCF     EQU     010DH
A78E           02070 CURS2    EQU     0A78EH   ;cursor update in ROM
0020           02080 SPACE    EQU     20
00E5           02090 VMODE    EQU     00E5
A75F           02100 TONGEN   EQU     0A75FH   ;play s/r in ROM
007F           02110 DEL      EQU     7FH
000A           02120 LF       EQU     0AH
0008           02130 BS       EQU     08
0007           02140 BL       EQU     07
000C           02150 FF       EQU     0CH
000E           02160 SO       EQU     0EH
000D           02170 CR       EQU     0DH
000F           02180 SI       EQU     0FH
00A0           02190 MEMTOP   EQU     00A0     ;memory-top loc in RAM
               02200
               02205                   ;REGISTER VALUES FOR 6545
               02210
3F12 5F        02220 L6545    DB      5FH      ;total horiz char count - 1
3F13 50        02230          DB      50       ;displayed chars per line
3F14 54        02240          DB      54       ;start of horiz sync
3F15 08        02250          DB      08
3F16 19        02260          DB      19       ;vert total char lines - 1
3F17 07        02270          DB      07       ;vert timimg fine adjust
3F18 19        02280          DB      19       ;no. of char lines on screen
3F19 19        02290          DB      19       ;start of vert sync
3F1A 48        02300          DB      48       ;display mode (non-interlace)
3F1B 0C        02310          DB      0CH      ;scan lines per char line
3F1C 6C        02320          DB      6CH      ;curs start and mode (flashing)
3F1D 0C        02330          DB      0CH      ;curs end
3F1E 00        02340          DB      00
3F1F 00        02350          DB      00
3F20 00        02360          DB      00
3F21 00        02370          DB      00
               02380
               02390
               02400
0000           02410          END
00000 Total errors
VMODE   00E5    ESCF    010D    L79C1   3F0B    SHIFT2  3F00
SUB1    3EED    SHIFT1  3EE4    CURS10  3EDF    TONGEN  A75F
DBL     3ED5    CRSINC  3EC7    DCR     3EE1    CLRLIN  3F09
LF1     3E98    DDL     3ECC    SCRST   F000    SCND    3E7A
CURS2   A78E    CURS1   3E6A    DSI     3EB1    SI      000F
DCR0    3EC0    CR      000D    DSO     3E97    SO      000E
FF      000C    DBL0    3ECA    BL      0007    DBS     3EC2
BS      0008    DLF     3EBA    LF      000A    DSP     3E92
SPACE   0020    DDL0    3E90    DEL     007F    SCRINT  3E45
SCREND  3E72    CLS     3E7F    CURSAD  010B    VDUCON  3E32
L6545   3F12    BEGIN   3E06    MEMTOP  00A0
```

■

Modems, Communications and the Bee

THE WORD 'modem' stands for MOdulator-DEModulator. Modems take the computer's digital signal and convert it into an analogue signal by coding on a carrier wave, and vice-versa. Mostly, modems are used to transfer data over public telephone lines, though they are often used for transfer over other media such as private lines or via radio.

Since telephones were designed in the pre-computer days they are best at carrying analogue signals (with a rather small bandwidth), thus limiting the speed at which data can be transferred.

There are two main types of modem – synchronous and asynchronous. We shall concentrate on the asynchronous types, since they are used for personal computer communication.

'Acoustic couplers' fit onto a standard phone and feed their signals into the mouthpiece of the phone. They are usually less reliable than 'direct-connect modems' which are connected directly to the phone lines. (*Warning*: It is illegal to connect any electrical equipment directly to telephone lines without Telecom permission.)

How do Modems Communicate?

Most modems use a principle known as 'frequency shift keying', or FSK for short. Basically, there is a carrier frequency which is transmitted constantly and is modulated up or down by 500 Hz to represent binary '1' or '0'. For example, if the carrier frequency is 1700 Hz, then 2200 Hz represents '1', and 1200 Hz represents '0'. Most phone lines have a frequency range (bandwidth) in the range 500 – 3000 Hz, which makes these FSK frequencies ideal for phone-line communication.

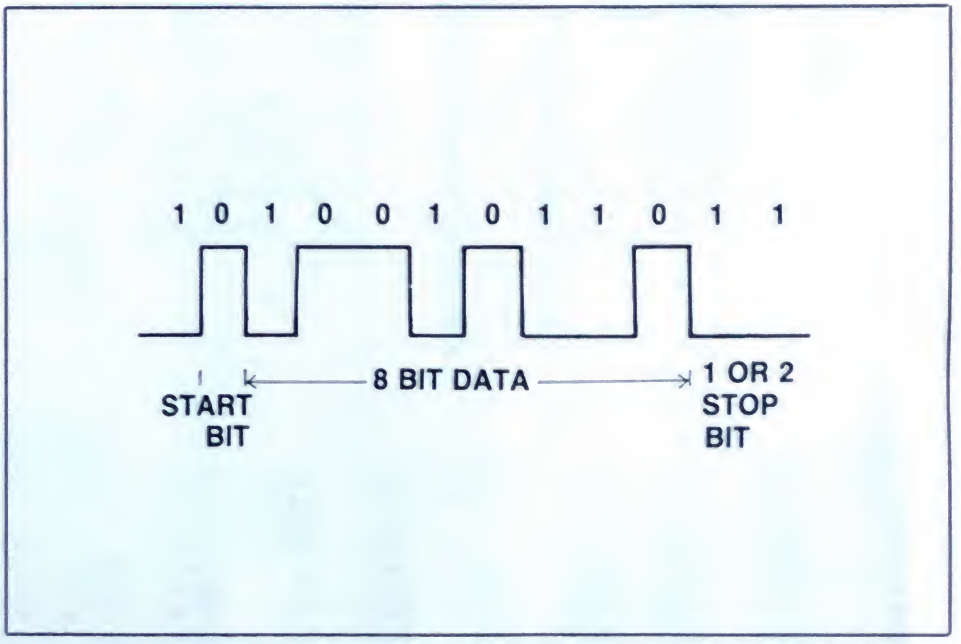


Figure 1. In an asynchronous transmission, each byte of data is a message in itself. A single 0 represents the start of the byte. This is followed by the data (seven or eight bits), an optional parity bit (not shown) and then one or two stop bits. When data is not being transmitted only the carrier frequency is transmitted by the modem, to show that the connection is still on line.

Humans are a sociable lot. They like communicating with one another and are always coming up with new ways of making the connection. For some time computer owners have had the option of communicating with others via their machines. For those who are thinking of buying a modem for their Bee, this article by Jon McCormack describes just what a modem is, how to connect it and what you can do with it once it's up and running.

So, now we know how to transmit the data, what is the format used? Unfortunately, this varies considerably and I have described one format below.

Asynchronous transmission means each character is sent individually, thus each byte sent has its own synchronising system (the data is transmitted serially – one bit at a time). This type of transmission is particularly suited to irregular transmission methods, such as directly from a keyboard. In effect, each character sent is a complete transmission. Figure 1 shows one mode for transmission of a single bit. The message always has one start bit, followed by the data, then either one or two stop bits. Sometimes a 'parity' bit is included. Parity has two types – odd parity and even parity – and is used to detect errors. With odd parity the number of 1s in the data byte is odd; even parity is where the number of 1s in the data byte is even.

These factors have an awful lot of possible combinations: one or two stop bits; odd, even or no parity; seven or eight bits per character, and so on. So what combination does everyone use? Well, the most common is no parity, one stop bit and eight bits per character. Most remote CP/M and bulletin board systems use this format, but don't count on this combination when you dial other, non-public systems.

Baud Rates

There is often confusion over what is actually meant by the word 'baud'. Baud rate is a measure of the speed of a line. It indicates the *maximum* number of changes on a line per second. So, if your modem has a speed of 300 baud and you have one start bit, eight data bits and one stop bit (a total of ten bits per character – no parity), then your modem transmits at a maximum speed of 30 characters per second. Most modems operate at this speed, while some operate at 1200 baud (which is often less reliable).

Data Transmission Arrangement

There are three types of transmission arrangement:

Simplex – sends data in one direction only.

Half duplex – transmits data in both directions, but one way at a time.

Full duplex – sends data in both directions at the same time.

See Figure 2 for details.

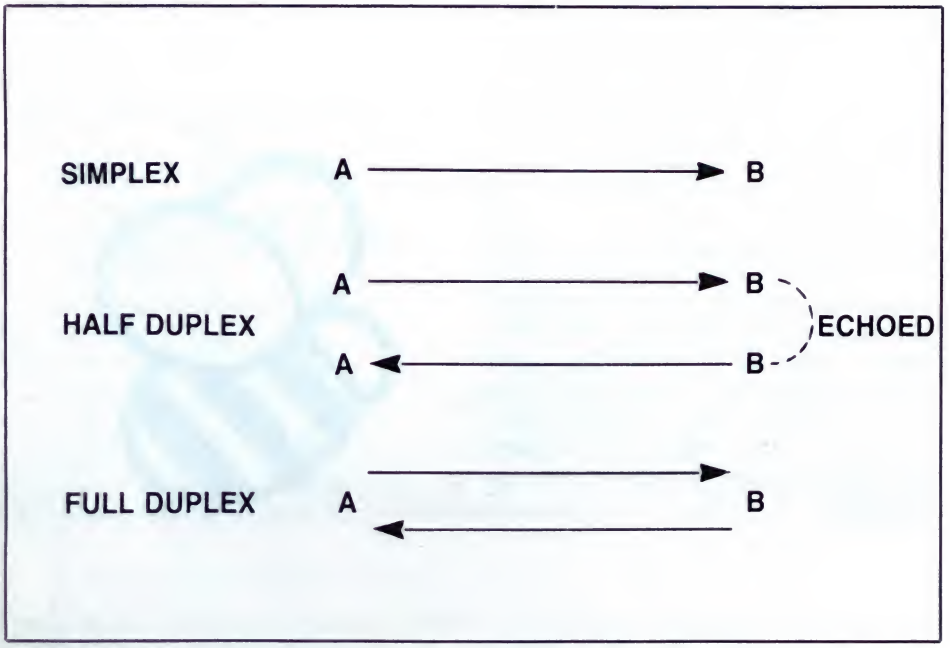


Figure 2. These are the three basic types of data transmission. With simplex, data only travels in one direction (as in a public address system, for example). In half duplex, data transfer is in both directions, but in only one direction at a time. Full duplex transmission allows data transmission in both directions simultaneously. Full duplex modems achieve this by having two carrier frequencies (one for receive, one for transmit) which are transmitted together. The line over which the transmission takes place must have sufficient band width to prevent the two frequencies from interfering with each other.

Modems, Communications and the Bee

Whether you choose full or half duplex depends on the machine you are talking to. Usually full duplex is chosen when talking to remote CP/M or bulletin board systems. When you want to talk direct to another computer owner, half-duplex is usually used. (Simplex transmission is rarely used in modem communications.)

The Microbee Modem

I bought the Microbee modem. (If you don't have a Microbee modem, don't despair – most of this discussion is relevant no matter what modem you have.) The biggest trouble I had was getting the modem itself. Applied Technology seems to pride itself on new product releases and heavy advertising of new peripherals, yet all too often these new products are not available. The Microbee modem was advertised some time ago, and only after several trips to the local Microbee shop did I manage to put in an order. Several weeks later the modem arrived.

As for software – I also purchased the new 'Telcom I' ROM package. This is a 4K communications package which comes with an instruction manual. The program is well written and caters for all the parameters described above (number of stop bits, parity, number of data bits, and full or half duplex). In addition, it provides 16 different baud rates ranging from 50 to 19,000 baud.

Telcom performs a number of functions:

1) Terminal Emulation: Typing 'full' or 'half' while in Telcom turns your Bee into a terminal for communication with other computers via the modem. The terminal emulated is the popular adm3A, which is used extensively with large, multi-user computers. In this mode the screen is 80 x 24 characters, just like in CP/M. While you're in the terminal mode you can do things like changing from half duplex to full duplex (or vice versa) and saving what appears on the screen to a Wordbee file (very handy) or directly to a printer. I found it best to save to Wordbee first, edit out the bits you don't want, and then get a printout.

2) File Transfer: Telcom also provides for transfer of BASIC, Wordbee and machine code files via the 'hobby standard' Ward Christensen protocol. This format provides error checking to see if the file came through, and is the same as that used by Modem7 and YAM (Yet Another Modem). The file is sent in 128-byte 'blocks', each accompanied by a checksum. If the checksum matches the value calculated at the receiving end, the next block is read; if not, the same block is sent again (up to ten times).

3) Real-Time Clock: A pseudo real-time clock is also provided, which shows the time at the top of the screen and has an alarm function. The clock is 'clocked' by a 50 Hz signal derived from the Microbee's crystal. This is fed to port B of the PIO chip. For me the clock was more trouble than it was worth, as the interrupts kept disrupting communications and writing stray characters all over the screen. I eventually removed the connection to the PIO chip.

Connection of the modem to my Microbee was straightforward. I just pulled out my old phone from its socket and plugged in the Beemodem (which comes with a phone so you can still talk to humans in the usual way). The other cable coming from the modem plugs into the RS232 port at the back of the Bee. It is good to see a product which requires no hidden extras; all the plugs are there and power is supplied from the Microbee.

Figure 3 shows different RS232 connections, depending on what you want to do with your modem and what options you use on the Telcom ROM. With everything installed, I was ready to 'talk' to the outside world.

Who Can You Talk To?

There are four main types of systems you can talk to:

a) Remote CP/M systems (RCPM)

b) Computer Bulletin Board Systems (CBBS)

c) Other modem owners

d) Mainframe computers and private installations.

Lets look at each of these in turn.

Remote CP/M systems are meant for CP/M users who wish to exchange software and information, however you don't have to have a CP/M system to log ▶

```
COMMON RS-232 CONNECTIONS:

PIN             FUNCTION
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
 2      Tx      Output signal from source (transmitter)
 3      Rx      Input signal to receiver
 5      CTS     Clear to Send (input) - transmitter should
                send if logic "YES"
 7      GND     GROUND voltage (Ø volts).
 9      1ØV     1Ø volt power supply (output)
2Ø       5V     5 volt power supply (output)
24      CLK     CLOCK (output) signal. Receiver is ready.

- - - - - - - - - - - - - - - - - - - - - - - - - - - -

CONNECTIONS:
        microbee to microbee:

    From Pin          To Pin
        2       ->      3
        3       ->      2
        5       ->     2Ø
        7       ->      7
       2Ø       ->      5

        microbee to modem:

     From pin         To Pin (on modem)
        2       ->      2
        3       ->      3
        5       ->      5
        7       ->      7
        9       ->      9
```

Figure 3. These are the different RS232 connections for transferring data from one Microbee to another, and from a Microbee to a modem. The configuration should be the same for other computers with the standard RS232 connections.

```
HOW MANY NULLS (Ø-9) DO YOU NEED? Ø
CAN YOUR TERMINAL DISPLAY LOWER CASE? Y

What is the name of Digital Research's standard debugger?_____

Welcome  to TARDIS.  This experimental system is designed to encourage  the
writing and  distribution  of  PUBLIC  DOMAIN CP/M software.

Any  suggestions or enquiries please leave a message for the SYSOP in MINIRBBS
or leave a comment when logging off.

(Prompting bell means system is ready for input).

What is your FIRST name ?jon
What is your LAST name ?mccormack
Checking user file...

Logging JON MCCORMACK to disk...

You are caller # :  5Ø22

Have you answered the user survey questions yet?y

Wait while I check to see if you have messages waiting ...

No, there aren't any messages for you, JON.
But check MINIRBBS anyway for public messages.


      *** Latest system news updated on 31-Jul-84 ***

Want latest system information before entering CP/M ?n

The time now is (Hrs:Mins:Secs).... 14:32:3Ø
You've been on the system for..... Ø0:Ø1:Ø6
```

Continued over the page . . .

Figure 4. A sample session with an RCPM system. The system first checks if your terminal requires nulls (delay between characters), and if it can display lower case. A password is then required ... and you're in. In this session I had a look at what files were available on drive A, and then went into the message sub-system to read a few of the messages. (User input is underlined.)

on. The first time you dial up you are asked some questions, such as the number of nulls to be transmitted (usually 0 – you only need more for slow terminals or printers) and if your terminal can display lower-case characters. Then you are asked a 'password' question, which is usually the name of Digital Research's standard debugger. You must answer this before you can proceed. (If you don't know the answer look at any book on CP/M.) The question is used because Telecom requires a password for anyone who wishes to leave messages on the system.

Once you're through all that, the fun begins. You are usually asked for your first and last names and the location you are calling from. You are then logged on to the system disk and put into the CP/M environment. From there you can execute most normal CP/M commands, or send and look at the messages from other users of the system. A sample output from a session with a CP/M system is shown in Figure 4 (this was obtained using the Wordbee 'Capture' function on the Telcom ROM).

CBB systems are similar to RCPMs, except they are designed for the exchange of messages between users. You usually have to pay a membership fee to join the system; though anyone can dial up and read the messages, only members can write them.

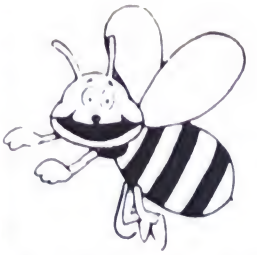
The standard set-up for RCPM and CBBSs is:

- 8 data bits
- 1 stop bit
- no parity
- full duplex, modem set to originate mode.

If you want to talk to other modem owners directly, you usually use half duplex. One of you is the ORIGINATor of the call and the other is the ANSWERer. Set the ANS/ORIG switch on your modem to one or the other (it is arbitrary who is who, as long as

... Figure 4 continued.

```
Entering CP/M...
A0>dir
AE$-SHOW.TXT   2k : BEETERM .OBJ   6k : BEEYAM  .OBJ  26k : BYE     .COM   6k
CHAT    .COM   2k : CPM     .HLP  38k : CPMCAT  .ALL  72k : CPV     .C     4k
CPV     .OBJ  10k : DIR     .COM   2k : DIR64   .COM   2k : DISASM  .BAS   8k
DISASM  .DOC   2k : ERAQ    .C     4k : ERAQ    .OBJ  10k : FIND    .COM   2k
FMAP    .OBJ   2k : FMAP3E  .ASM  12k : FORMAT  .OBJ   2k : HELP    .COM   2k
IBM-MOD7.ASM   6k : IBM-MOD7.CØM   8k : IBM-MOD7.DOC   2k : KERMTCPM.DIF   4k
LUNAR   .DOC   2k : LUNAR   .MWB   2k : MEDTERM .DOC   8k : MEDTERM .OBJ   6k
MESSAGES.HLP   8k : MINIRBBS.COM  26k : MKBAUD  .MAC   2k : MKBAUD  .OBJ   2k
MODEM   .HLP  14k : MS      .WS    4k : NAME    .C     2k : NAME    .OBJ   6k
NSWEEP  .OBJ   8k : ONLINE  .COM   2k : ONLINE64.COM   2k : ORBIT297.TAB   2k
PKTADDR .DOC   4k : QUICK   .HLP   2k : SIGMCAT .ALL 146k : SOFTWARE.HLP  12k
STERM   .DOC  20k : SW-READ .ME    2k : SWEEP41 .DOC  18k : SWEEP41 .OBJ  28k
SWINIT41.OBJ  14k : TARCAT  .ALL   8k : THIS-SYS.HLP  16k : TYPE    .COM  12k
UNSCRUB .C     6k : UNSCRUB .OBJ   8k : WHATSNEW.COM   2k : WILDEXP .C     8k
XYAM    .COM  28k : XYAMHELP.T     2k : YAM             2k : YAMPHONE.T     2k

A0>minirbbs

TARDIS Remote CP/M Message Subsystem......

Active # of msg's:  63
You are caller # :  5022
Next msg # will be: 1800

Function [E,R,S,K,C,G,P,X,Q,T,B (or '?' if not known)]??

Functions supported:
S--> Scan messages        R--> Retrieve message
E--> Enter message        K--> Kill message
P--> Prompt (bel) togl    X--> eXpert user mode
Q--> Quick summary        C--> Comment before exit to CP/M
G--> Go direct to CP/M    T--> Time on system
B--> Read bulletin on latest software

Commands may be strung together, separated by semicolons.
For example, 'R;123' retrieves message # 123.
For forward sequential retrieval, use  '+' after Msg #.

Function [E,R,S,K,C,G,P,X,Q,T,B (or '?' if not known)]?r
MSG # ( 1 - 1799 ) to retrieve (C/R to end)?1798+
Use Ctl-S to Pause, Repeated Ctl-K's to Abort.

Msg #  1798  Date entered: 04/08/84  From: GREG BLACK
To: SYSOP  About: REQUEST

Recently I downloaded YAMMBEE.OBJ from this system. Now I
would like to obtain the file YAMHELP.T which it apparently
needs to run its Help function. I would also like to obtain
the source for the YAM program. If these are available (I
did not find them with FIND) I would like to get them. I hope
this is clear and that you can help me. Thanks.  ...Greg.


Msg #  1799  Date entered: 04/08/84  From: SIMON GERRATY
To: ALL  About: BEE SOFTWARE?

HAS ANYONE GOT ANY TIPS ON HOW TO DOWN LOAD SOFTWARE TO
A MICROBEE WHICH IS NOT RUNING CP/M?
I HAVE A 32K IC WITH NETWORK ROM FITTED.

                                THANKS...
                                      SIMON

** End of messages **

Function [E,R,S,K,C,G,P,X,Q,T,B (or '?' if not known)]?g
Character count:   2917 typed by system -  8 typed by you.

The time now is (Hrs:Mins:Secs).... 14:36:08
You've been on the system for..... 00:04:44


Now, back to CP/M...
```



you're not both the same). You're then ready to transfer files via the Telcom commands SAVE and LOAD.

Communication with mainframes and minicomputers is also possible – I have used my modem to operate university mainframe computers remotely. You can also phone up special databases in the United States (if you can afford the phone bill!). Note that for communication with these systems you must have a special user code and password, which are assigned to you by whoever runs the installation. It is unwise (and inconvenient) to phone up private installations and go experimenting.

Conclusions

I have some comments to make on the Beemodem and modems in general. Most modems run at 300 baud, which is very slow. This speed is fine if you just want to read a few messages or copy small files, but when you want to transfer large files it takes a lot of time. Some of the more expensive modems have a 1200-baud option which is a lot faster, however, all the systems I know of only work at 300 baud. The only value in buying a 1200-baud modem is for communicating with a fellow computer user who also has a 1200-baud modem.

The Beemodem and Telcom ROM set are good partners, and so far have worked well for me. They put worldwide data communications within reach of any home computer owner, for a reasonable cost. I would recommend buying the Telcom ROM set, especially as it provides almost all the software necessary for practical communications.

When you dial up RCPM and CBBS systems, please answer all the questions properly to help the system operator (sysop) make improvements for the users of the system. Also, don't leave stupid messages or stay logged on for a long time – give all users a fair go.

Remote CP/M and Computer Bulletin Board Systems

Australia

Software Tools RCPM (ST-RCPM): (07) 378 9530 24 hours EST.
Your Computer BBS (MiCC-BBS): (02) 662 1686 24 hours EST.
Micro Design Lab RCPM (MDL-RCPM): (02) 663 0151 24 hours EST.
Sydney Public Access RCPM (SPA-RCPM): (02) 808 3536 24 hours EST.
Omen RTRS (OM-RTRS): (02) 498 2495 1630-0900 + 24 hours weekends.
Sydney TRS-80 UG RTRS (STRUG-RTRS): (02) 332 2494 24 hours EST.
Prophet BBS (PROPHET-BBS): (02) 628 7030 24 hours EST.
Dick Smith Electronics BBS: (02) 887 2276 24 hours EST.
Newcastle RCPM (NCLE-RCPM): (049) 68 5385 1700-0830 + 24 hours weekends.
Melbourne CBBS (MICOM-CBBS): (03) 762 5088 24 hours EST.
Sorcerer CBBS (SCUA CBBS): (03) 836 4616 24 hours EST.
TARDIS RCPM (TARDIS-RCPM): (03) 67 7760 1800-0800 + 24 hours weekends.
PC Connection IBBS (PCC-IBBS): (03) 528 3750 24 hours EST.
Gippsland RCPM (GL-RCPM): (051) 34 1563 24 hours EST.
Gippsland MAIL BUS (GL-MBUS): (051) 27 7245 24 hours EST.
Adelaide Micro UG BBS (AMUG-BBS): (08) 271 2043 1000-2200 CST.
Computer Ventures BBS (CV-BBS): (08) 255 9146 24 hours CST.
Outback RCPM (OUTB-RCPM): (089) 27 7111 24 hours CST.
OMEN II RTRS (OM2-RTRS): (089) 27 4454 24 hours CST.
OMEN III RTRS (OM3-RTRS): (09) 279 8555 0800-2400 + 24 hours weekends.

New Zealand

Attache RBBS (ATT-RBBS): ISD 64 9 78 9084 + 24 hours NZT Domestic (09) 76 9084.

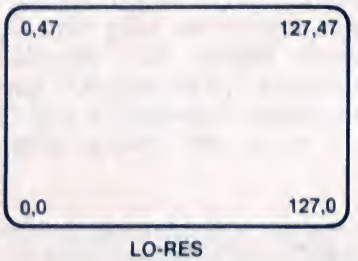
Figure 5. Please note the operation times. Most installations also require a 'password' before you can access or write messages to the system. To obtain a password, you sometimes have to pay a membership fee.

Figure 5 gives a list of most of the systems you can dial up in Australia. One thing you find after purchasing a modem is the dramatic increase in your phone bill. Keep those STD and overseas calls to a minimum! One might compare the use of modems now to the CB radio craze of a few years ago. Whether or not the home data communications craze will flop as CBs did remains to be seen. However, at the moment I can recommend anyone who is thinking of buying a modem to do so. It will release you from the isolation of your single-user computer, and allow you to 'talk' to thousands of other computer users around the world. Perhaps I'll see one of your messages on an RCPM or CBBS system soon. ■

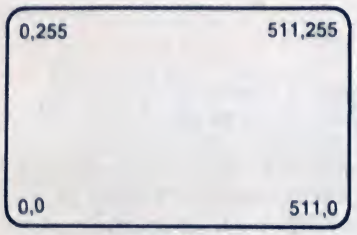
Microbee Graphics

Most reviews of the Microbee comment at some stage on the limitations of its high-resolution graphics. However, many other computers do not even have such high-resolution, and the Microbee provides an alternative low-resolution as standard. The aim of this article, by Mike Oborn, is therefore to correct false impressions of the Microbee's graphics commands of Microworld BASIC.

IN LOW-RES mode the VDU is divided into 48 rows of 128 dots, which increases to a resolution of 256 rows of 512 dots in high-res mode. Figure 1 depicts these layouts.



LO-RES



HI-RES

Figure 1.

Unlike some other computers, the Microbee has the origin of its screen display in the bottom left-hand corner.

In both high-res and low-res modes, groups of dots can be addressed in the form of character spaces. Figure 2 depicts these character spaces in both high-res and low-res modes, where it can be seen that low-res characters are far more chunky.

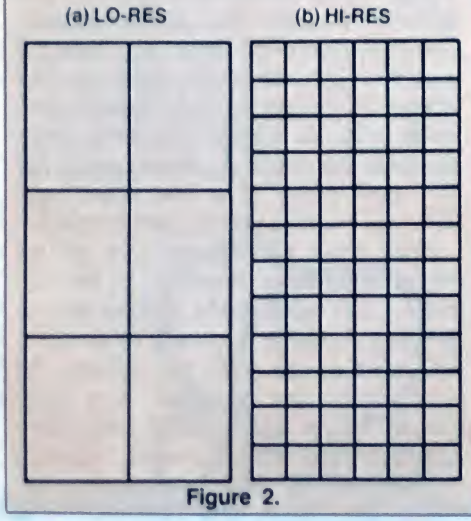


Figure 2.

A quick calculation (on your Microbee?) will reveal that in either format there are 16 rows of 64 character spaces on the VDU, a total of 1024 addresses in screen memory. This layout is depicted in Figure 3, where the screen origin is now depicted in the top left-hand corner.

Figure 3.

A character space can be addressed in either of two ways – by reference to its x and y co-ordinates, or by reference to its character position, which starts at 0 in the top left-hand corner and increases to 1023 in the bottom right-hand corner. This leads us to study the range of commands provided in Microworld BASIC for graphics displays.

Graphics Commands

There are two ways of placing graphics on the VDU. The first is by turning on (or off) individual dots to form the desired shape; the commands for this are SET, RESET, INVERT, PLOT and POINT. Each of these commands is expressed in terms of the x and y co-ordinates, as depicted in Figure 1, for example SET x,y, or PLOT x1, y1 TO x2, y2. These commands can be used in either high-res or low-res mode provided the values of x and y fall within the respective parameters of that mode.

For compatibility with programs written for computers with their screen origins in the top left-hand corner, the letter 'H' can be appended to these command words to invert the y-axis: RESETH x, y. For this reason a space must follow the command word if a suffix is not added. There are other suffixes which can be added to the PLOT command: 'I' inverts the state of the dots being addressed, and 'R' resets the dots.

The following short program will demonstrate the use of some of these commands:

```
100 CLS: LORES
110 FOR X = 20 TO 100 STEP 4: SET X,10: SET X,40: NEXT X
120 FOR X = 10 TO 40 STEP 4: SET 20,X: SET 100,X: NEXT X
130 FOR J = 1 TO 1000: NEXT J
140 PLOTI 20,10 TO 20,40 TO 100,40 TO 100,10 TO 20,10
150 NORMAL : END
```

This program SETs dots in the form of a rectangle in the middle of the screen; only each fourth dot is SET. The PLOTI command then follows the sides of the rectangle, inverting the state of each dot and so giving a rectangle with three out of each four dots SET. The command CLS clears the VDU screen. If hi-res mode is used it is not necessary to clear the screen as this command does it automatically.

Line 130 is only a timing loop to clearly separate the two phases of the screen display. (NB: A graphics mode (like 'LORES') must be specified before any of the graphics commands covered here are used.)

The NORMAL command returns the PRINT output to normal format, which is not strictly necessary except after INVERSE, UNDERLINE and PCG.

While it may be desirable at times to program a display by addressing individual dots, it is also possible to address groups of dots in the form of the character spaces depicted in Figure 2. The

second method of placing graphics on the VDU is by addressing these character spaces with the following commands: POKE, PEEK, CURSor and PRINT.

First the CURSor command. This takes two forms – CURS x or CURS x,y. The first form of this command is similar to the PRINT AT command used in some other dialects of BASIC. The value of x is that for the position starting at 0 in the top left-hand corner of Figure 3. Alternatively the CURSor can be positioned by referring to its x and y co-ordinates. Once the cursor is positioned we can PRINT the required graphics detail to the VDU. Both the following lines will print the same graphics character (the letter A) in the same place:

120 CURS 960: PRINT CHR$(65)

or

120 CURS 1,16: PRINT CHR$(65)

By the use of the POKE command we can place data into the RAM area of the Microbee which holds its screen memory; this starts at address 61440. Position 0 on Figure 3 is this screen memory address; position 1 is 61441 and so on to position 1023, which is memory address 62463. Thus the following line will also print the same graphics character in the same place as either of the two lines above:

120 POKE 62400,65

Now we know how to display our chosen graphics shapes we had better consider the range of shapes available.

Graphics Characters

In Figure 2 we saw that the character spaces in both low-res and high-res modes comprise a series of dots, each one individually addressable. High-res graphics use the Microbee's programmable character generator (PCG), a detailed study of which will be left until the second part of this article. However, normal ASCII-coded characters are also formed within the same 8 by 16 character space of figure 2(b), but the top four rows are left blank and the bottom three rows are only used for descenders on characters such as p and q and the underline in UNDERLINE mode

This means that most ASCII characters are formed within rows 12 down to 4 – the first column (actually bit 07)

CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL
0		32		64	@	96	`	128		160		192		224	
1		33	!	65	A	97	a	129		161		193		225	
2		34	"	66	B	98	b	130		162		194		226	
3		35	#	67	C	99	c	131		163		195		227	
4		36	$	68	D	100	d	132		164		196		228	
5		37	%	69	E	101	e	133		165		197		229	
6		38	&	70	F	102	f	134		166		198		230	
7		39	'	71	G	103	g	135		167		199		231	
8		40	(	72	H	104	h	136		168		200		232	
9		41	)	73	I	105	i	137		169		201		233	
10		42	*	74	J	106	j	138		170		202		234	
11		43	+	75	K	107	k	139		171		203		235	
12		44	,	76	L	108	l	140		172		204		236	
13		45	-	77	M	109	m	141		173		205		237	
14		46	.	78	N	110	n	142		174		206		238	
15		47	/	79	O	111	o	143		175		207		239	
16		48	0	80	P	112	p	144		176		208		240	
17		49	1	81	Q	113	q	145		177		209		241	
18		50	2	82	R	114	r	146		178		210		242	
19		51	3	83	S	115	s	147		179		211		243	
20		52	4	84	T	116	t	148		180		212		244	
21		53	5	85	U	117	u	149		181		213		245	
22		54	6	86	V	118	v	150		182		214		246	
23		55	7	87	W	119	w	151		183		215		247	
24		56	8	88	X	120	x	152		184		216		248	
25		57	9	89	Y	121	y	153		185		217		249	
26		58	:	90	Z	122	z	154		186		218		250	
27		59	;	91	[	123	{	155		187		219		251	
28		60	<	92	\	124	:	156		188		220		252	
29		61	=	93	]	125	}	157		189		221		253	
30		62	>	94	^	126	~	158		190		222		254	
31		63	?	95	_	127		159		191		223		255	

Table 1.

CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL
0		32		64	@	96	`	128		160		192		224	
1		33	!	65	A	97	a	129		161		193		225	
2		34	"	66	B	98	b	130		162		194		226	
3		35	#	67	C	99	c	131		163		195		227	
4		36	$	68	D	100	d	132		164		196		228	
5		37	%	69	E	101	e	133		165		197		229	
6		38	&	70	F	102	f	134		166		198		230	
7		39	'	71	G	103	g	135		167		199		231	
8		40	(	72	H	104	h	136		168		200		232	
9		41	)	73	I	105	i	137		169		201		233	
10		42	*	74	J	106	j	138		170		202		234	
11		43	+	75	K	107	k	139		171		203		235	
12		44	,	76	L	108	l	140		172		204		236	
13		45	-	77	M	109	m	141		173		205		237	
14		46	.	78	N	110	n	142		174		206		238	
15		47	/	79	O	111	o	143		175		207		239	
16		48	0	80	P	112	p	144		176		208		240	
17		49	1	81	Q	113	q	145		177		209		241	
18		50	2	82	R	114	r	146		178		210		242	
19		51	3	83	S	115	s	147		179		211		243	
20		52	4	84	T	116	t	148		180		212		244	
21		53	5	85	U	117	u	149		181		213		245	
22		54	6	86	V	118	v	150		182		214		246	
23		55	7	87	W	119	w	151		183		215		247	
24		56	8	88	X	120	x	152		184		216		248	
25		57	9	89	Y	121	y	153		185		217		249	
26		58	:	90	Z	122	z	154		186		218		250	
27		59	;	91	[	123	{	155		187		219		251	
28		60	<	92	\	124	:	156		188		220		252	
29		61	=	93	]	125	}	157		189		221		253	
30		62	>	94	^	126	~	158		190		222		254	
31		63	?	95	_	127		159		191		223		255	

Table 2.

is also left blank to provide a space between each character on a line. Using this portion of each character space ensures adequate spacing between lines of text, but makes it difficult to build a picture using the character set provided. A complete listing of the graphics symbols using the POKE command in high-res mode is provided in Table 1, where it can be seen that 32 special graphics symbols (codes 0 to 31) are provided.

Using the PRINT CHR$(x) command (where x is the ASCII code between 0 and 225) will produce a slightly reduced listing. Codes 0 to 31 are not printable; they are control characters – for example CHR(7) is BELL – Control G; Code 127 is delete. By the way, ASCII code 95 is not included on the Microbee's keyboard, but can be generated by pressing the shift and delete keys simultaneously.

From Table 1 it can be seen that in high-res mode, codes 128 to 255 print inverse characters of codes 0 to 127.

Table 2 provides another listing of graphics characters, this time in low-res using the POKE command. Where codes 129 to 255 were inverse characters in high-res mode, these have now become chunky graphics. However, note that codes 192 to 255 are merely repeats of codes 128 to 191, created by numbering each of the six dots that comprised the low-res character spaces of Figure 2(a). This has been reproduced in Figure 4.

Figure 4.

If we want to turn on the dots numbered 1, 2, 4, 16 and 32, for example, we total the value of the relevant dots, add them to 128 and we have the code for that shape specified in Table 2. Unlike high-res and other ASCII characters, low-res characters fill the full character space and can be joined to build solid pictures. Note that the high-res characters of codes 0 to 31 are also available in low-res mode. Using the PRINT CHR(x) command produces the same reduced character range as we saw above with high-res.

PCG and the Five Graphics

With the standard graphics behind us, it is now time to consider the role of the programmable character generator (PCG) in creating the Microbee's five graphics modes. This piece of hardware not only controls the high-res and low-res modes, but also UNDERLINE, INVERSE and PCG, which we have not yet considered. It is this PCG which is responsible for the Microbee's graphics limitations.

In the first part of this article we saw that high-res graphics characters (including the normal ASCII characters) are formed in a character space of eight dots across by 16 down, as shown in Figure 5. Each dot (or bit) is represented by a number between 1 and 128, with the value of each row (one byte in the Microbee's memory) being represented by a number between 0 and 255 – 0 if no bit is to be displayed, or up to 255 if all dots in that row are to be displayed.

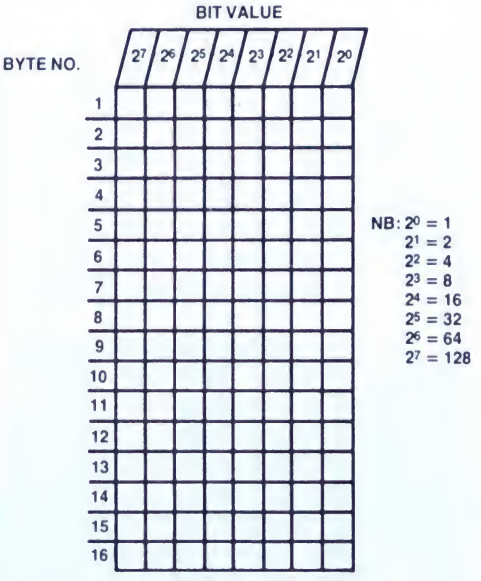


Figure 5.

Each PCG character can therefore be represented in memory by 16 bytes, each of which represents a combination of dots that is to be displayed in each row to form the desired shape. With 128 characters available in the PCG, it takes 2048 bytes (2K) to represent these characters.

The required bytes are POKEd into memory using DATA and READ commands. PCG memory starts at memory location 63488, with the first character occupying bytes 63488 to 63503. The following program demonstrates the technique of loading a PCG character into the first character space in the PCG RAM. Figure 6 shows how the DATA statement was calculated.

The program:

```
100 DATA 56,56,56,26,254,186,186,
186, 186,56,40,40,40,40,40,40
  110 FOR I = 63488 TO 63503
  120 READ A: POKE I,A
  130 NEXT I
  140 PCG: CLS
  150 POKE 61440,128
  160 NORMAL
```

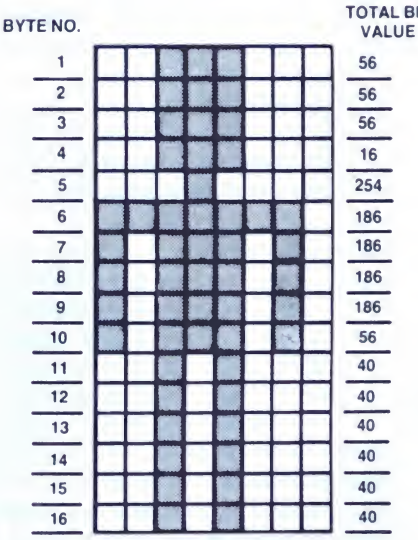


Figure 6.

In line 150 we have used the POKE command to place the PCG character in the first location in screen memory (remember the top left-hand character space?). The reason for this choice is that we cannot PRINT the first 32 ASCII characters (remember?) and therefore must resort to the POKE command. However, when using the POKE command we must add 128 to the ASCII code because the PCG characters occupy the second 128 characters in PCG RAM. Nevertheless, with the PRINT command, PCG characters are coded 0 to 127 – clear?

The following program will demonstrate this by using the POKE command to print the PCG RAM contents in the top four rows of the screen. Initially the inverse character set occupies the PCG RAM where your own PCG characters will be loaded, and these are the second 128 characters listed. (NB: If you have not reset your Microbee since using another graphics mode, the PCG RAM will contain those characters, not the INVERSE set.)

```
100 CLS
110 PCG
120 FOR X = 0 TO 255
130 POKE 61440 + X,X
140 NEXT X
150 NORMAL
```

By substituting the following line in the above program, the difference in the PCG RAM using the PRINT command can be seen – the inverse set is listed first.

```
130 CURS 0 + X: PRINT CHR$(X)
```

Now we have seen how PCG characters are created, it is time to use them in graphics displays.

```
00100 CLS: LORES
00110 FOR X=20 TO 100 STEP 4:SET X,10:SET X,40: NEXT X
00120 FOR X=10 TO 40 STEP 4: SET 20,X: SET 100,X: NEXT X
00130 FOR J=1 TO 1000: NEXT J
00140 PLOTI 20,10 TO 20,40 TO 100,40 TO 100,10 TO 20,10
00150 NORMAL: END
```

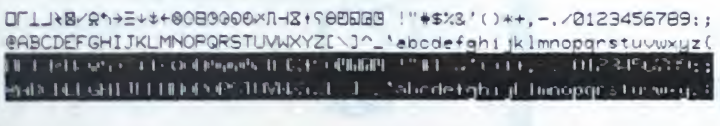
```
00100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
00110 DATA 0,0,0,0,0,0,7,8,17,18,17,16,8,8,8,8,16
00120 DATA 0,0,0,0,128,192,160,28,8,176,32,64,128,128,64,32,16
00130 DATA 0,0,0,1,30,8,6,1,6,5,8,4,7,8,0,0
00140 DATA 16,32,192,0,0,2,9,4,134,122,34,199,0,0,0,0
00150 DATA 16,16,16,32,64,128,0,128,64,48,8,4,184,64,0,0
00160 FOR I=63488 +65*16 TO 63488+71*16-1
00170 READ A: POKE I,A
00180 NEXT I
00190 CLS:PCG
00200 X=1
00210 CURS X,8: PRINT "ABC"
00220 CURS X,9:PRINT "DEF"
00230 NORMAL
00240 END
```

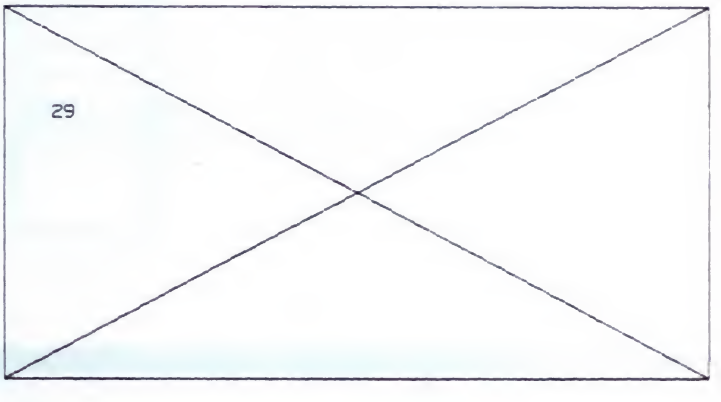
Listing 1 (above) produces the duck shown on the left. Below and left are other samples of the Microbee's graphics.

```
00100 DATA 56,56,56,16,254,186,186,186,186,56,40,40,40,40,40,40
00110 FOR I= 63488 TO 63503
00120 READ A: POKE I,A
00130 NEXT I
00140 PCG: CLS
00150 POKE 61440,128
00160 NORMAL
```

```
00100 HIRES
00110 PLOT 0,0 TO 0,255 TO 511,255 TO 511,0 TO 0,0
00120 PLOT 0,0 TO 511,255
00130 PLOT 0,255 TO 511,0
00140 CURS 5,5: PRINT USED
00150 END
```

```
00100 CLS
00110 PCG
00120 FOR X=0 TO 255
00130 POKE 61440+X,X
00140 NEXT X
00150 NORMAL
```





Microbee Graphics

PCG Graphics

We have already used the POKE, CURS and PRINT commands to place PCG characters on the VDU. The other command used is PEEK.

While there is a limit of 128 programmable characters, there is no limit on the number of times any of these characters can be placed on the VDU to build up a picture – in other words it could be reproduced 1024 times to fill each character space on the VDU! Also, a larger character can be built up over several character spaces – either side by side or in rows. The program in Listing 1 will draw a duck which extends over six character spaces – three in row 8 and three in row 9 immediately below the row 8 characters.

In line 160 we have POKEd the DATA into ASCII codes 65 to 70 (remember it could have been 33 to 38, when the PRINT characters would have been "!"#& %&", for example). Movement can be achieved by a FOR NEXT loop increasing the value of X to a maximum of 61 (the end of the line). Character 'A' created in the program has all bits off – it is a blank.

To create the impression of movement it is necessary to remove the first character from the VDU and replace it with another character in the next character space. This can be achieved in one action by ensuring the overlaid PCG character covers the first character while still moving one character space. By adding a trailing blank ('A') to each PRINT command ('AABC' and 'ADEF') we will ensure that none of the earlier duck is displayed.

The movement created by the above program will not look smooth. To improve this, the PCG character which overlays the earlier one should provide some natural movement (for example in the legs and tail). The smaller the movement, the more PCG characters, but also smoother graphics. Additionally, moving less than one character space each time will smooth out the graphics display but will require even more PCG characters to handle this movement. All that is needed is patience, plenty of graph paper and of course a maximum of 128 PCG characters.

High-Res Limitations

Now that we have a better understanding of the PCG characters, we can consider how the programmable character generator creates high-res graphics.

To demonstrate, the following program draws a line around the VDU screen and then joins the diagonally opposite corners. The command USED in line 140 returns the number of PCG characters used in PLOTting this display – 29 in all.

```
100 HIRES
110 PLOT 0,0 TO 0,255 TO 511,255
TO 511,0 TO 0,0
120 PLOT 0,0 TO 511,255
130 PLOT 0,255 TO 511,0
140 CURS 5,5: PRINT USED
150 END
```

For there to have been only 29 PCG characters used, even though the lines PLOTted in this program must have entered around 280 character spaces, means that where the PCG character created by the PLOTted line is a duplicate of another PCG character used earlier, a new PCG character is not generated, but the first PCG character is reused. This can be demonstrated by PEEKing into the screen memory at any location and seeing what PCG character has been created for that character space. The following can be input in the IMMEDIATE mode to return the ASCII code of the PCG character created.

```
PRINT PEEK(61444)
```

The ASCII code returned was 131. If once again in the IMMEDIATE mode we input:

```
PRINT CHR$(131)
```

the output is one character space of the top line of the VDU showing that segment of the PLOTted line.

It is possible to run out of PCG characters quite quickly drawing circles or by PLOTting random lines. Therefore to reduce the number of PCG characters used in high-res mode, either restrict your graphics to a portion of the screen or draw straight lines where possible, rather than angled lines or curves.

Other Graphics Modes

We still have not covered the UNDERLINE and INVERSE modes. These also use the PCG RAM and are more easily explained now that we have covered the other modes more fully.

When using the UNDERLINE command the last byte of each character loaded into the PCG RAM is coded 255 - remember, all dots turned on? This means all characters will have the bottom line in that character space turned on (that is, an underline). *All* output is underlined until the command NORMAL is encountered.

Under the INVERSE command, as the normal ASCII characters are loaded all bits are inverted. *All* output is then in inverse video until once again the NORMAL command is encountered.

Conclusion

This completes our look at the Microbee's graphics. Obviously the choice of the programmable character generator provides some limitations on the Microbee's graphics, but it does allow finer resolution of graphics displays. ■

More Bee Graphics

After reading Mike Oborn's article 'Microbee Graphics' when it was originally published in 'Your Computer' magazine, John Martin was inspired to work on unearthing more of the Bee's graphics capabilities. Here he describes how to create another set of graphics characters.

AS YOU must know if you read Mike Oborn's graphics article on pages 22-26 of this publication, the normal Microbee screen contains 1024 characters, distributed in 16 rows, with each row containing 64 characters. In the 'low-res' graphics mode, each character is composed of six 'dots' – two across by three up – each of which can be 'on' or 'off'. This gives 64 (2 raised to the power of 6) possible low-res characters. The Programmable Character Generator (PCG) contains room for 128 characters, and in low-res mode each of the 64 characters is repeated. However, when using the low-res mode we are not aware of these individual characters. The final effect is of a screen of 128 dots across by 48 dots up, where the bottom left corner is addressed as (0,0) and the top right corner as (127,47).

Now, what would happen if we wanted each dot to have three modes – 'off', 'on' or 'half-on' (that is, shaded)? If we kept six dots per character, we would need 729 (3 raised to the power of 6) characters – too many to fit in the PCG. So we would have to reduce the number of dots per character; down to four, in fact, which would give us 81 characters.

Figure 1.

Figure 1 shows how a character can be divided into four dots. Each dot position is given a number, starting with 1 at the top, and multiplying by 3 to give 3, 9, and 27 at the bottom. Each dot mode is given a value: 0 for off, 1 for half-on, and 2 for on. Thus, a character whose four dots were half-on, on, off, and half-on (see Figure 2) would occupy a position in the PCG given by:

1x1 + 2x3 + 0x9 + 1x27 = 34

Since each character is divided into four dots vertically, the screen will be able to hold 64 dots across and 64 dots up. So there are fewer dots available than in low-res (4096 compared with 6144), however, the dots can have three modes instead of only two and are a nice square shape. ▶

More Bee Graphics

Chomp, Chomp, Chomp, Chomp

The characters are composed of four byte patterns: 'Off' dots are (in binary)

00000000 = 0 decimal,

on dots are

11111111 = 255 decimal,

and 'half-on' dots are

01010101 = 85 decimal

alternating with

10101010 = 170 decimal.

So to produce these graphics characters, we need a program which will put 0 in the PCG for 'off' dots, 255 for 'on' dots, and 85 and 170 alternately for 'half-on' dots.

Figure 3 contains a BASIC subroutine which will produce these characters in the PCG from character 0 to character 80. A machine code routine contained in the DATA statements is poked into memory from location 336 onwards. Line 1140 runs this routine, and the remainder of the subroutine produces four 'cursors' in characters 81 to 84 for use in a later subroutine (4000).

If you type in this subroutine, preceded by a calling program ...

```
10 GOSUB 1000
20 END
```

... RUN it and then type:

```
>PCG:PRINT CHR(34):NORMAL
```

you should see the character in Figure 2 displayed on the screen.

Figure 3.

```
01000 REM...Graphics Characters Generated...
01010 RESTORE 1050
01020 Z=336
01030 READ Y:IF Y<0 THEN 1140
01040 POKE Z,Y:Z=Z+1:GOTO1030
01050 DATA 33,0,248,221,33,0,0,6,4,221
01060 DATA 35,221,54,16,4,16,248,221,33,4
01070 DATA 0,14,4,221,126,16,254,4,204,152,1,254
01080 DATA 2,204,160,1,254,1,204,171,1,221,43,13
01090 DATA 32,233,221,33,4,0,167,221,203,16,30
01100 DATA 48,216,221,54,16,4,221,43,221,229
01110 DATA 193,62,0,185,32,235,201,6,4,54,0
01120 DATA 35,16,251,201,6,2,54,85,35,54,170,35
01130 DATA 16,248,201,6,4,54,255,35,16,251,201,-1
01140 Z=USR(336)
01150 FOR I=64784 TO 64847
01160 POKE I,0
01170 NEXT I
01180 FOR I=64785 TO 64845 STEP 20
01190 POKE I,24:POKE I+1,24:NEXT I
01200 RETURN
```

The Routine Work

Having created our graphics characters, we now need to be able to use them! Unfortunately, we can't just use SET, PLOT, and so on, but we can write some of our own routines to do similar things. Figure 4 contains a subroutine which will set a dot, whose position is (I,J), in mode Z. I is the position across the screen, J is the position up the screen, where the bottom left corner is (0,0) and the top right corner is (63,63). Z is 0 for off, 1 for half-on, and 2 for on. The calling command is GOSUB (I,J,Z) 2000. Note that putting Z=0 makes this command similar to the BASIC RESET.

Figure 2. Represents a character whose dots are half-on, on, off, and half-on.

Figure 4.

```
02000 REM...SET I,J in mode Z...
02010 VAR(I,J,Z)
02020 IF I<0 OR I>63 THEN RETURN
02030 IF J<0 OR J>63 THEN RETURN
02040 A=62400+I-J/4*64
02050 B=4-J+J/4*4
02060 C=PEEK(A)
02070 IF C<128 THEN LET C=0 ELSE LET C=C-128
02080 D=1:IF B>1 THEN FOR E=1 TO B-1:D=D*3:NEXT E
02090 E=C/D
02100 E=E-E/3*3
02110 C=C+(Z-E)*D+128
02120 POKE A,C
02130 RETURN
```

Figure 5 contains a subroutine to draw a line from point (V,W) to the point (X,Y) in mode Z. The calling command is GOSUB (V,W,X,Y,Z) 3000. Again, when Z=0 this command is similar to the BASIC PLOTR.

Note the BASIC commands INVERT and PLOTI do not have equivalents here, because of the three available modes.

Figure 5.

```
03000 REM...PLOT V,W TO X,Y in mode Z...
03010 VAR(V,W,X,Y,Z)
03020 T=INT(SGN(FLT(X-V)))
03030 U=INT(SGN(FLT(Y-W)))
03040 IF U=0 AND T=0 THEN GOSUB[X,Y,Z]2000:RETURN
03050 A=(Y-W)*U:B=(X-V)*T
03060 IF A>B THEN 3100
03070 FOR I=V TO X STEP T
03080 J=(Y-W)*(I-V)+(X-V)*U/2:J=J/(X-V)+W
03090 GOSUB[I,J,Z]2000:NEXT I:RETURN
03100 FOR J=W TO Y STEP U
03110 I=(X-V)*(J-W)+(Y-W)*T/2:I=I/(Y-W)+V
03120 GOSUB[I,J,Z]2000:NEXT J:RETURN
```

Figure 6.

```
00010 GOSUB 1000
00020 CLS
00030 Z=1
00040 FOR Q=0 TO 63 STEP 3
00050 Z=1-(Z=1)
00060 GOSUB[Q,Q,Q,63,Z]3000
00070 GOSUB[Q,63,63,63-Q,Z]3000
00080 GOSUB[63,63-Q,63-Q,0,Z]3000
00090 GOSUB[63-Q,0,0,Q,Z]3000
00100 NEXT Q
00110 GOSUB 6000
00120 END
```

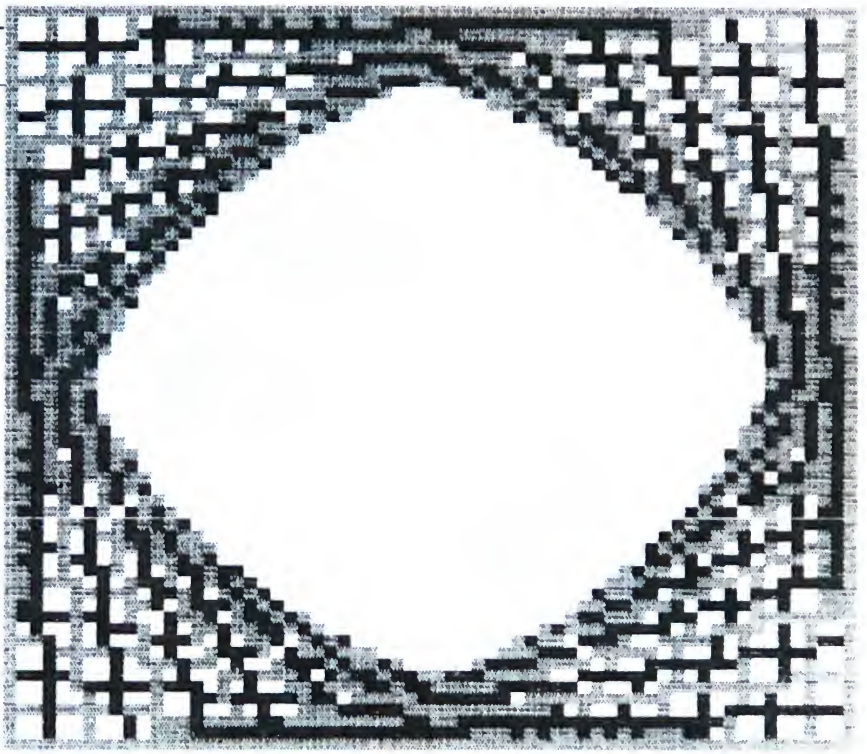
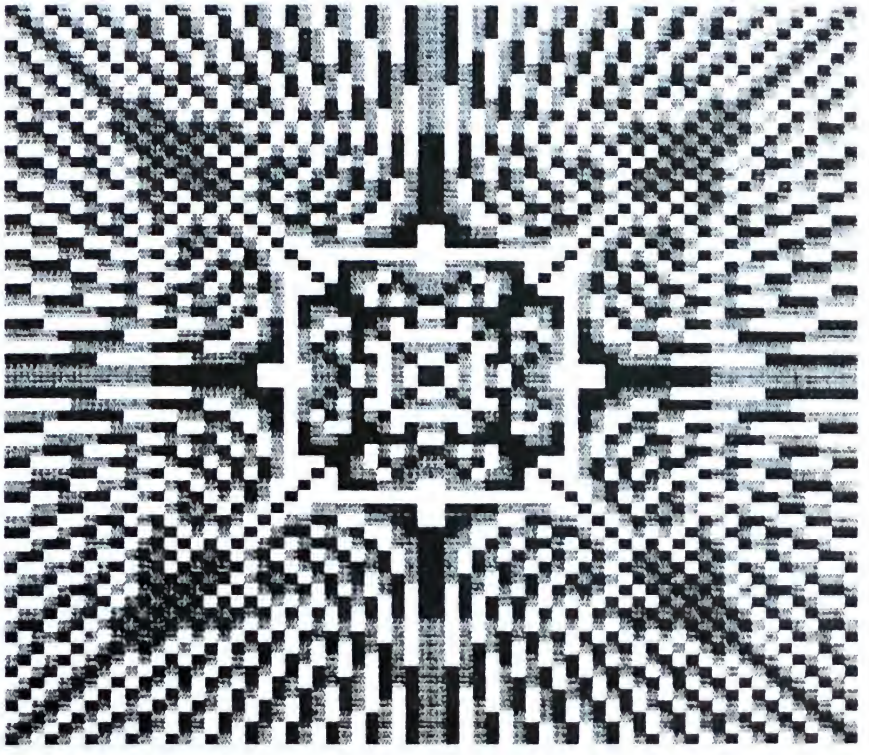


Figure 7.

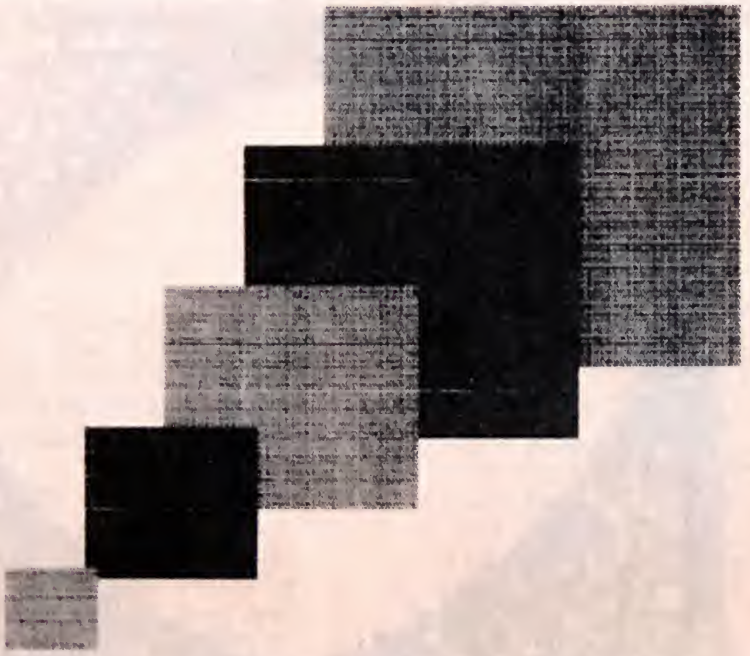
```
00010 GOSUB1000
00020 CLS
00030 FORQ=0TO31
00040 Z=(Q+1)-(Q+1)/3*3
00050 GOSUB[0,31-Q,63,32+Q,Z]3000
00060 GOSUB[0,32+Q,63,31-Q,Z]3000
00070 GOSUB[31-Q,0,32+Q,63,Z]3000
00080 GOSUB[32+Q,0,31-Q,63,Z]3000
00090 NEXTQ
00100 GOSUB6000
00110 END
```



In Figures 6 and 7 there are a couple of programs which call on this line-drawing routine, together with the results obtained. The reference to subroutine 6000 is to a suitable 'screen dump' routine. Note also in line 50 of figure 6 the method used to 'toggle' the line mode between 'on' and 'half-on'. ▶

Figure 8.

```
05000 REM...Fill a Rectangle...
05010 VAR(V,W,X,Y,Z)
05020 T=INT(SGN(FLT(X-V)))
05030 U=INT(SGN(FLT(Y-W)))
05040 IF T=0 OR U=0 THEN GOSUB[V,W,X,Y,Z]3000:RETURN
05050 FOR I=V TO X STEP T
05060 FOR J=W TO Y STEP U
05070 GOSUB[I,J,Z]2000
05080 NEXT J
05090 NEXT I
05100 RETURN
```



■

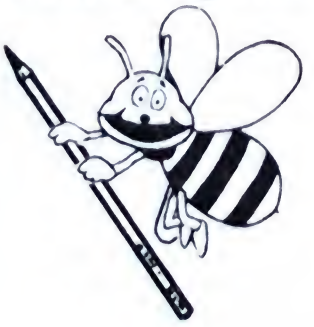
Figure 9.

```
00010 GOSUB 1000
00020 CLS
00025 Z=2
00030 FOR P=30 TO 0 STEP -6
00040 Z=1-(Z=1)
00050 V=P:W=P:X=2*P:Y=2*P
00060 GOSUB[V,W,X,Y,Z]5000
00070 NEXT P
00080 GOSUB 6000
00090 END
```

Figure 10.

```
04000 REM...Cursor Control...
04010 I=31:J=31:Z=2:A=61983:B=1:C=130
04020 POKE A,208+B:FOR D=1 TO 8:NEXT D:POKE A,C
04030 Y=PEEK(258):IF Y=255 THEN 4020
04040 IF Y=5:RETURN
04050 IF Y=3:CLS:GOTO 4020
04060 IF Y=16:GOSUB 6000:GOTO 4020
04070 IF Y>31 AND Y<35:Z=Y-32:GOTO 4120
04080 IF Y=9 OR Y=15 OR Y=21:J=J-(J<63)
04090 IF Y=13 OR Y=14 OR Y=44:J=J+(J>0)
04100 IF Y=10 OR Y=14 OR Y=21:I=I+(I>0)
04110 IF Y=11 OR Y=15 OR Y=44:I=I-(I<63)
04120 GOSUB[I,J,Z]2000
04130 GOTO 4020
```

More Bee Graphics



Another useful routine to have, not directly available in BASIC, is one to draw blocks. Figure 8 contains a subroutine which fills in (in mode Z) a rectangle whose diagonally opposite corners are at (V,W) and (X,Y). Figure 9 is a program using this to produce a 'projection' effect.

Finally, we have in Figure 10 a subroutine which allows the user to draw or modify screen patterns with a 'cursor'. Memory location 258 contains a number which tells which key is pressed at any time (regardless of upper or lower case) – if no key is pressed, it contains 255. This is checked in line 4030. Line 4020 flashes the cursor. The cursor is moved up, left, right or down by pressing I, J, K, or M (notice the pattern formed by these keys on the keyboard). Pressing U, O, N or < moves the cursor up and left, up and right, and so on. The mode is changed by pressing 0, 1 or 2. Pressing C clears the screen, pressing P will print out the screen at any time, and pressing E will return to the controlling program. Notice in lines 4080 to 4110 the method used to keep the cursor on the screen. Figure 11 shows an application of this; the 'block' subroutine was first called four times and then the letters were added with the 'cursor' subroutine.

Well, there it is – another set of graphics for the Microbee! Certainly different to any of the others, and a lot of fun to investigate with many pleasant patterns to be discovered. They even have some practical applications (designing Fairisle sweaters?). But, because the routines are in BASIC, they are *very slow* to run! If anyone out there has the time and skill to spare, and can speed things up by redoing these routines in machine code, write to *Your Computer* or *ETI* magazines at 140 Joynton Avenue, Waterloo 2017. ■

A Handy Place For Machine Language Routines

There are literally hundreds of small, interesting machine code routines being published for the Microbee. Many address the difficulty encountered in selecting a place to store code where it won't be either wiped out by BASIC or itself wipe out BASIC's own essential data (scratch, pointers, and so on). John Dowdall has found a handy place for your machine language routines.

WHEN THE COMPUTER is a 64K Microbee, you don't want to garble BASIC itself by putting a routine in BASIC RAM storage. There is at least one place available for this purpose which not only doesn't interfere with BASIC or any of its data, but is totally invisible to BASIC under ordinary circumstances. There are, of course, some circumstances under which BASIC will both 'see' and corrupt your routine, but these can generally be avoided.

Let's sidetrack just for a moment to consider what happens when BASIC finds a program line containing the keyword 'REM'.

If you check your Microbee Handbook (page 120 in the new edition), you will see that BASIC will ignore *everything* after the REM during execution of the program, even to the extent of leaving lower case text unchanged.

Have you leaped ahead of me yet? Suffered a blinding flash of inspiration? Heard BELs ringing?

If not, stay with me for a few moments while I explain. The fact that BASIC ignores anything after a REM means that it will *also* ignore a short (less than 180 bytes) machine-code routine stored there.

There is only one remaining difficulty – how to get the machine code in there? It would be easy to do if the keyboard had 256 keys – one for each possible single byte number – you'd simply type them in! As it turns out it is nearly that easy. The way BASIC stores a REM line in program memory is: two bytes for the line number, one byte for the line length, and then the text of that line followed by a byte containing 13. The next byte after the 13 is for the NEXT line number.

In the line:

10 REM This is a comment line

the stored codes are: 0 10 27 32 161 32 84 104 105 115 32 105 115 32 97 32 99 111 109 109 101 110 116 32 108 105 110 101 13. That fifth byte, 161, is how BASIC stores REM, and the numbers following it are the ASCII codes for the rest of the line.

So, down to the method (I know, you were wondering ...). If you make the first line of your program the one in which you store the routine, the addresses to POKE the data are easy to find. A BASIC program is stored from address 2304 decimal onwards. Allowing two bytes for the line number, one for the length, one more for a space and one for the REM, address 2309 will be where to begin POKEing your routine.

Create the REM line, and after the REM, place at least as many asterisks as your routine length.

```
10 REM**********
**********
20 REM The rest of your program from here on
```

Add this routine to the end of your program:

```
25000 FOR I=0 TO XX : REM Replace XX with routine length
25010 READ A:POKE2309+I,A
25020 NEXT I
25030 DATA 46,0,89,80,62,248,211,2,21,32,253,62,184
25040 DATA 211,2,29,32,253,45,32,237,201
```

and type GOTO 25000 <RETURN>.

When you get the >__ prompt, list line 10 – most if not all of the asterisks have gone. You can now get on with your USR calls and the rest of the program. I usually add the M/L routines after the main program is finished and completely de-bugged.

It is advisable to make the REM line (in which you save M/L code) the first line in your program. If however, you add, delete, or edit lines prior to this REM line, *remember* to change the start address of your POKEs in line 25010.

Once your routine has been installed and the lines from 25000 deleted, you can now SAVE or SAVEF the program quite safely and retrieve it with the machine code intact.

So there you are; a neat and safe way to save your machine code. The example code used in the DATA statements is Milan Hudacek's Sound Effects Tone Generator, described elsewhere in this magazine. ■

Solving Equations on the Bee

If your work or interests require you to solve more than the occasional equation, Brian Syms has come up with three methods of using the Bee to help you nut them out.

MANY EQUATIONS crop up in science and engineering, which can't be solved directly. A simple example would be to find the value of 't' for which

t – exp(-t)

Solutions of equations of this type usually involve some kind of trial and error, in which you try to guess a possible solution and then substitute that guess into the equation. The result of this substitution is then used to refine the guess, and the second guess is then substituted, and so on. The process is known as 'iteration' and we talk about 'finding the root of the equation'. Obviously, computers are ideal for this sort of repeated calculation.

There are several simple techniques for solving equations by computer. They all start with rewriting the equation so that when the variable has the appropriate value the equation will be equal to zero. For all other values of the variable the equation will not be zero. In other words, instead of writing

t = exp(-t)

we write

f(t) = t – exp(-t)

and we try to find the value of 't' for which f(t) is zero.

My equation-solving methods all require initial guesses or 'seeds'. In many cases we already have some idea, from the problem, what sort of answer to expect. If we don't know where to look for a solution the computer can be used to step through values of 't' to look for the values between which f(t) changes sign. The root must be near these values. Figure 1 shows such a listing for our example problem. It can be seen that the answer, or root, lies between 0.5 and 0.6.

The Binary Search Method

One simple but effective technique for solving such equations is called the 'Binary Search Method'. This is very similar to the old number guessing game – the one where you try to guess a number, someone tells you if your guess is too big or too small, and eventually you zero in on the secret number.

We need two 'seeds', t1 and t2, one larger and one smaller than the accurate solution, so that f(t1) is greater than zero and f(t2) is less than zero. The average

t3 = (t1 + t2)/2

will therefore be closer to the root. We then calculate f(t3) and if it is greater than zero we replace t1 with t3, otherwise we replace t2 with t3. We can now calculate the average of the two new guesses, and go around the loop again and again until the difference between t1 and t2 is less than some predetermined value.

Figure 2 shows a BASIC listing of a program to solve our example by the Binary Search Method, together with the result of running the program. In order to illustrate the answer better the program prints out all the guesses.

Figure 1.

```
00100 REM figure 1, table of t1
      and f(t1). 15/4/84
00110 PRINT " t          f(t)"
00120 FOR T1= 0 TO 1 STEP 0.1
00130 F0 = T1 - EXP(-T1)
00140 PRINT T1,[F8.2  F0]
00150 NEXT T1
00160 END


>run

 t          f(t)
 0.        -1.00
 0.1       -0.80
 0.2       -0.61
 0.3       -0.44
 0.4       -0.27
 0.5       -0.10
 0.6        0.05
 0.7        0.20
 0.8        0.35
 0.9        0.49
 1.         0.63
```

Figure 2.

```
00100 REM 15/4/84 version
00110 PRINT"  Binary Search method":PRINT :PRINT
00120 INPUT "First Seed:";T1
00130 INPUT "Second Seed: ";T2
00140 IF ABS(T1 - T2)<1E-4 THEN 200
00150 T3 = ( T1 + T2)/2 : REM next guess
00160 PRINT [F8.4 T3];
00170 F0 = T3 - EXP(- T3)
00180 IF F0 <0 THEN LET T1 = T3: GOTO140
00190 T2 = T3 : GOTO 140
00200 PRINT: PRINT" it worked ! the answer is at t = ";[F8.4 T3]
00210 END

>run

  Binary Search method


First Seed: .5
Second Seed:  .6
  0.5500  0.5750  0.5625  0.5687  0.5656  0.5671  0.5664  0.5667
  0.5669  0.5670
it worked ! the answer is at t =   0.5670
```

The Secant Method

Another way of achieving the same result is known as the 'secant' method. This also requires two seeds; the difference this time is that it doesn't matter whether or not the two guesses straddle the root or not. The idea is that if we plot t1,f(t1) and t2,f(t2), they can be used to point to the value of t,(t3) at which f(t) is zero. Figure 3 gives some idea of how this works. Because f(t) isn't a linear function of t, t3 won't be the exact root. It should, however, be closer than t1 or t2.

Using similar triangles we can show that

$$t3 = (f(t1)t2 - f(t2)t1) / (f(t1) - f(t2))$$

If we now replace t1 with t3 and t2 with t1, we can go around the loop again and again until t1 is sufficiently close to t2. Figure 4 shows a listing to solve our problem by this method, including a sample run. Once again, all the intermediate guesses have been printed out for us to inspect.

Newton's Method

There is one other technique commonly employed to find the solutions to equations. This is known as 'Newton's' method, presumably after Sir Isaac. It has the advantage of only needing one seed and the disadvantage of being very difficult to describe why it works. If our first guess is t1, we calculate

$$g(t1) = (f(t1 + d) - f(t1)) / d$$

where

$$d = t1/1e5.$$

(Readers with a knowledge of calculus might recognise this as a crude form of numerical differentiation.)

Our second guess is then given by:

$$t2 = t1 - f(t1) / g(t1)$$

This process can be repeated time and again until two successive guesses are equal. A program to solve our example by this technique is shown in Figure 5.

By now you might be wondering why we need three methods; why not be content with one? The sad fact is that they don't always work! Sometimes one of the methods will fail to find the root. For my own use I have strung the three programs together in a menu so I can try each one in turn. If two of them agree on an answer, I can be pretty confident that it is correct. ■

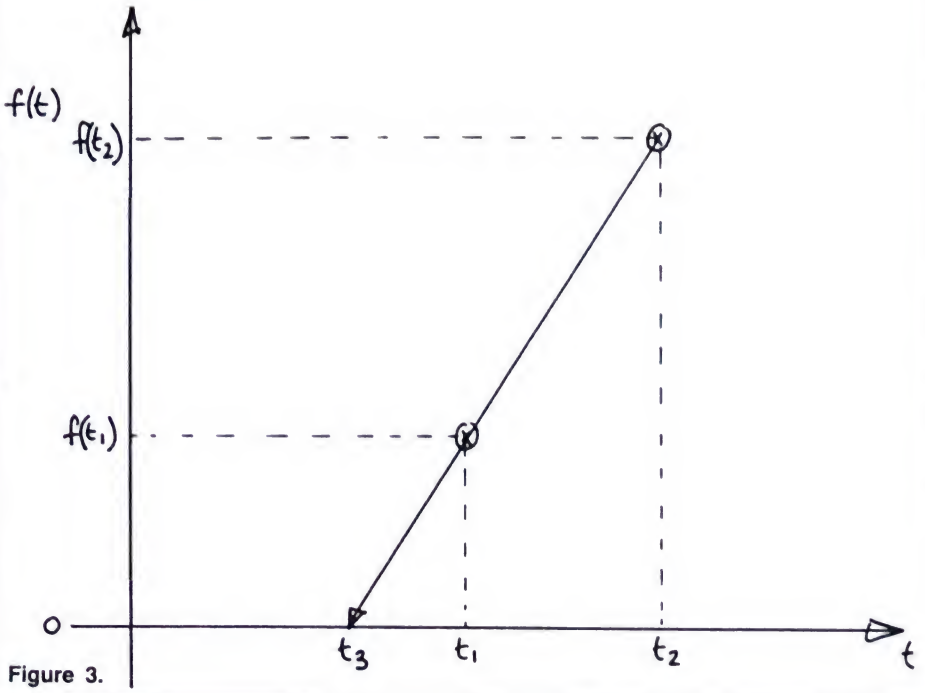


Figure 3.

Figure 4.

```
00100 REM 15/4/84 version
00110 PRINT"  Secant Method": PRINT :PRINT
00120 INPUT "First Seed: ";T1
00130 INPUT "Second Seed: ";T2
00140 F1 = T1 - EXP(-T1)
00150 F2 = T2 - EXP(-T2)
00160 T3 = (F1*T2 - F2*T1)/(F1 - F2):REM next guess
00170 PRINT[F8.4 T3];
00180 IF ABS(T1-T2)<1E-4 THEN 230
00190 T2 = T1
00200 T1 = T3
00210 IF ABS(T1-T2)<1E-4 THEN 230
00220 GOTO 140
00230 PRINT:PRINT " Success again! the root is at t = ";[F8.4 T3]
00240 END
>run
  Secant Method
First Seed:  .5
Second Seed:  .6
  0.5675  0.5671  0.5671
 Success again! the root is at t =   0.5671
```

Figure 5.

```
00100 REM 15/4/84 version
00110 PRINT "Newton's Method":PRINT
00120 INPUT " Seed = ";T1
00130 D0 = T1/1E5
00140 T0=T1
00150 GOSUB 260
00160 F1=F0
00170 T0= T1 + D0
00180 GOSUB 260
00190 G1 = (F0 - F1)/D0
00200 T2 = T1 - F1/G1
00210 PRINT[F8.4 T2];
00220 IF ABS(T1-T2) <1E-4 THEN 240
00230 T1 = T2: GOTO 130
00240 PRINT:PRINT"Hooray for Sir Isaac! the root is
      at t = ";[F8.4 T1]
00250 END
00260 F0 = T0 - EXP(-T0)
00270 RETURN
>run
Newton's Method
 Seed =  .5
  0.5670  0.5671  0.5671
Hooray for Sir Isaac! the root is at t =   0.5671
```

Microbee Music

By Milan Hudacek

This article explains the principles of Microbee sound generation and explores some unusual application possibilities that you won't find in the user's manual.

THE MICROBEE tone generator uses a single bit of port B of the programmable input/output interface. The speaker is connected to the output bit by a driving transistor. Clearly, there's not much hardware involved – that's why all the sound-generation relies upon software.

The obvious shortcomings of this approach are twofold: You can generate only single-voiced melody, and you cannot do anything else while playing, because the central processing unit (CPU) is fully occupied with handling your speaker bit. The actual tone generation is done simply by periodic setting and resetting of the speaker bit. Generally speaking, the faster you set and reset, the higher the pitch of the tone.

The shape of the resulting audible signal is approximately a square wave, as shown on this figure:

The frequency (pitch) of the tone is the reciprocal value of the period T, which is the sum of the Ton and Toff intervals:

frequency = 1/T = 1/(Ton + Toff)

For example, to generate the piano tone 'middle C', the required frequency is approximately 262 Hz and Ton plus Toff = 1/262 = 3.8 milliseconds.

The quality (timbre) of the tone is given by the Ton and Toff intervals ratio. The MicroWorld BASIC Play command always uses 1:1 ratio but we can experiment and create some interesting effects using variable ratios as well.

The loudness of your MicroBee is fixed and, unfortunately, cannot be changed by program independently from the timbre. (However, if you'd like to alleviate somewhat the roar of your machine, I recommend replacing of the resistor H27 in series with your speaker. Its original value being 27 ohm, I replaced it with a 100 ohm resistors and the headache ceased miraculously. The R27 resistor is located just under the right shift key.)

Now, let's try practically what has been said above – periodic setting and resetting of the speaker bit using the MicroWorld BASIC out command. The out command has this format:

OUT int1,int2

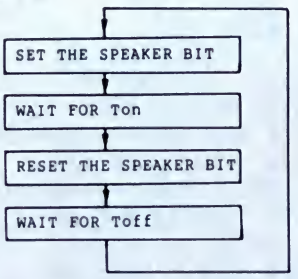
Int1 and int2 are integer expressions: int1 represents the output port address, and int2 is the data which should be output.

The address of your port B is 2, but what data should actually be output? If you don't use your RS232 interface, you can output 0 and 255 alternatively, toggling all eight bits simultaneously (255 decimal corresponds to 11111111 binary). If your printer or whatever is connected to your RS232 line, then the output data will have to affect only the speaker bit (D6) and not the RS232 output or DTR control line. The fastest way to find out about the "quiescent" state of the B port is to type in:

PRINT IN (2)

Your MicroBee probably responds with 184, which is the decimal equivalent of 10111000 binary. Setting the D6 bit (they are counted from right to left, starting with zero), we get 11111000 binary, which corresponds to 248 decimal. Thus, by alternating your port B output between 184 and 248, you toggle your speaker bit without your serial-connected printer going mad.

Now, try a simple program using the above described principle. The program will do this:



If the Ton and Toff intervals are preset, see what happens:

```
00100 INPUT "N,F : ";N,F
00110 OUT 2,248
00120 FOR I=1 TO N
00130 NEXT I
00140 OUT 2,184
00150 FOR I=1 TO F
00160 NEXT I
00170 GOTO 110
```

After entering the run command, you have to input the N (Ton) and F (Toff) values. You can enter any number between 1 and 255, but even if you try N,F – 1 (the shortest interval possible), you'll find to your great disappointment that what you hear is a low-frequency clapping, instead of the expected music of spheres. (You should have expected that! Didn't you know that BASIC is too slow for audio-frequency handling?)

Notice that when you press any key on your keyboard, the clapping frequency lowers. This is because the BASIC interpreter checks, after each line of program, whether you've pressed the BREAK key (which you have to use to exit from our program). Whenever you press a key, you include a short sub-routine comparing the key code with the BREAK code – this prolongs the delays, and in turn lowers the frequency.
But don't despair. There exists an easy way to overcome the laziness of your BASIC – we shall discuss it later. Meanwhile, let's create some modest sound-effects with what we have learned so far. Here is a couple of them:

```
00100 REM RADIATION DETECTOR
00110 REM (PERHAPS SUITABLE FOR YOUR THREE MILE ISLAND GAME)
00120 IF RND(.5 THEN 150
00130 OUT 2,248
00140 GOTO 120
00150 OUT 2,184
00160 GOTO 120
```

```
00100 REM HEARTBEAT
00110 OUT 2,248:OUT 2,184:OUT 2,248
00120 FOR I=1 TO 250
00130 NEXT I
00140 OUT 2,184
00150 FOR I=1 TO 250
00160 NEXT I
00170 GOTO 110
```

```
00100 NAME-IT-YOURSELF-1
00110 FOR N=1 TO 30
00120 OUT 2,248:OUT 2,184
00130 FOR I=1 TO N
00140 NEXT I
00150 NEXT N
00160 GOTO 110
```

```
00100 REM NAME-IT-YOURSELF-2
00110 FOR N=1 TO 40
00120 GOSUB 180
00130 NEXT N
00140 FOR N=40 TO 1 STEP -1
00150 GOSUB 180
00160 NEXT N
00170 GOTO 110
00180 OUT 2,248:OUT 2,184
00190 OUT 2,248:OUT 2,184
00200 OUT 2,248:OUT 2,184
00210 FOR I=1 TO N
00220 NEXT I
00230 RETURN
```

Now, putting Assembler to work: The only way to implement audio-frequency bit-handling in microcomputers is to program the appropriate sub-routine in machine code, by using your MicroBee Editor/Assembler ROM. However, you don't have to own it to benefit from our "research", or even understand Assembly language, if you want to incorporate our sound-generating sub-routine into

your BASIC program application.

The sound-generating sub-routine, to be most universal, has to fulfill these requirements:

1. Ton and Toff intervals must be independently presettable.
2. Tone duration must be presettable (but not necessarily independently).
3. The sub-routine must be able to be called from BASIC, using the USR function. Moreover, it should be relocatable to fit anywhere without the need of reassembling.

The USR function implemented in MicroWorld BASIC has this format: integer variable = USR (integer expression 1, integer expression 2).

The integer variable on the left is compulsory and denotes the fact that USR is a function and not a command. In our case, it will be a dummy variable because we don't need any value to be passed back from our sub-routine to BASIC.

The integer expression 1 is the address of the user program. We have three possible places to put our sub-routine:

1. Hires scratch pad (first 100 bytes from 0)
2. BASIC program area (see text)
3. Spare RAM (1024 bytes from 62464 decimal)

You cannot use the Hires scratch pad if you're going to use high-resolution graphics in your application program

You can use your "spare RAM" anytime if you don't mind disturbances on the screen during sound-generation – this is due to the hardware construction of your MicroBee and cannot be avoided. (The "spare RAM" is, in fact, an unused half of the PCG character RAM. When it's accessed, the VDU switches its internal data bus to the CPU data bus and cannot refresh the monitor screen. This isn't apparent normally if you just POKE or PEEK data in this area. Running programs here, however, causes trouble.)

Placing the sub-routine in your BASIC program area (2304 to 16383 for 16 kilobytes, 2304 to 32767 for the 32-kilobyte system) is dangerous – precautions have to be made for the sub-routine not to collide with your BASIC program and variables on one side and your strings and stack on the other. However, this might be the only way if you wanted to use Hires graphics as well.

(To be quite honest, we have yet another possible place for our sub-routine: in the mysterious illegal area under 2304 decimal. Part of it is used as a BASIC scratch pad, but there do exist gaps; probably 512-1023, 1112-1280, and so on. But don't tell anyone.)

In our initial experiments, let's place our sub-routine in the Hires scratch pad, starting with the address 0. The integer expression 2 of the USR function is passed to the machine code sub-routine in the BC register pair. In fact, it's optional, but we certainly have use of it. What about passing the Ton delay interval in B and Toff in C registers?

The remaining parameter to be passed is the tone duration. We shall pass it by POKEing it into a known address. Here is our long-awaited tone-generating sub-routine:

```
0000            00100           ORG 0
0000 2E00       00110           LD L,0          ;TONE DURATION (DUMMY VALUE)
0002 59         00120 LOOP0:    LD E,C          ;LOAD 'OFF' INTERVAL COUNT
0003 50         00130           LD D,B          ;LOAD 'ON' INTERVAL COUNT
0004 3EF8       00140           LD A,248        ;'SPEAKER BIT ON' VALUE
0006 D302       00150           OUT (2),A       ;OUTPUT TO PORT B
0008 15         00160 LOOP1:    DEC D           ;DECREMENT COUNT
0009 20FD       00170           JR NZ,LOOP1     ;LOOP FOR 'ON' INTERVAL
000B 3E08       00180           LD A,184        ;'SPEAKER BIT OFF' VALUE
000D D302       00190           OUT (2),A       ;OUTPUT TO PORT B
000F 1D         00200 LOOP2:    DEC E           ;DECREMENT COUNT
0010 20FD       00210           JR NZ,LOOP2     ;LOOP FOR 'OFF' INTERVAL
0012 2D         00220           DEC L           ;BUMP TONE LENGTH COUNTER
0013 20EF       00230           JR NZ,LOOP0     ;LOOP FOR NEXT PERIOD
0015 C9         00240           RET             ;RETURN TO BASIC
0000            00250           END
00000 Total errors

LOOP2   000F    LOOP1   0008    LOOP0   0002
```

The sub-routine is relocatable, as it uses only relative jumps.

The tone-duration value will have to be POKEd to the second byte of the sub-routine, which is the operand byte of the LD L, data instruction. (This is a very simple and innocent example of self-modifying code, and computer-science purists are already going to burn us at the stake. Nevertheless, I think that, considering our simple application, this is the most efficient approach.)

The sub-routine is quite short and the most convenient way to use it in a BASIC program is to use the DATA command and POKE it into the working area, at the beginning of your BASIC program.

It's now necessary to convert the hexadecimal object code produced by the Assembler into decimal. I've done it to spare you the toil, so all you have to do is just to type in:

```
00100 DATA 46,0,89,80,62,248,211,2,21,32,253,62,184,211,2,29,32,
253,45,32,237,201
00110 FOR I=0 TO 21
00120 READ D
00130 POKE I,D
00140 NEXT I
00150 INPUT "B,C,L : ";B,C,L
00160 POKE 1,L
00170 I=USR(0,B*256+C)
00180 GOTO 150
```

Notice how the Ton and Toff parameters are passed: integer expression 2 = Ton*256 = Toff. This is because we want to pass Ton in register B and Toff in register C. Multiplying by 256 decimal virtually equals to shifting by eight binary bits.

Now you can experiment yourself. After starting with the RUN command you have to enter B (Ton interval length), C (Toff interval) and L (tone length). All values should be entered in the range of 1-255 plus zero, which is actually equivalent to 256.

Exit from the program by hitting the BREAK key. Notice that:

1. The timbre of the tone changes if you change the B/C ratio, keeping B+C constant.
2. The tone duration does not depend on L only but on B+C as well. In fact, it's proportional to L*(B+C) because L is actually the number of tone periods.

You can already use this sub-routine for some nice sound-effects but there still remains a lot to be improved and a lot to be discovered. Meanwhile, as usual, some interesting effects for your perusal. Here are seven of them, combined in a single program. If typed in carefully and started by RUN command, it should produce amazing sounds until stopped by the BREAK key.

```
00100 REM ***** MICROBEE SOUND EFFECTS *****
00110 DATA 46,0,89,80,62,248,211,2,21,32,253,62,184,211,2,29,32,
253,45,32,237,201
00120 FOR I=0 TO 21
00130 READ D
00140 POKE I,D
00150 NEXT I
00160 REM ***** RANDOM EFFECT SELECTION *****
00170 ON INT(RND*8) GOSUB 200,250,350,420,480,540,600
00180 GOTO 170
00190 REM ***** EFFECT 1 *****
00200 A=INT(RND*30)
00210 T=INT(RND*100)
00220 FOR N=0 TO 2*T STEP INT(RND*20)+1
00230 M=INT(ABS(FLT(N-T)))
00240 POKE 1,A
00250 I=USR(0,(T+1-M)*256+M+1)
00260 NEXT N
00270 RETURN
00280 REM ***** EFFECT 2 *****
00290 FOR N=1 TO INT(RND*20)+1
00300 POKE 1,INT(RND*100)
00310 I=USR(0,INT(RND*256)*256+INT(RND*256))
00320 NEXT N
00330 RETURN
00340 REM ***** EFFECT 3 *****
00350 T=INT(RND*100)
00360 FOR N=1 TO T STEP INT(RND*10)+1
00370 POKE 1,INT(RND*100)
00380 I=USR(0,(T-N+1)*256+T-N+1)
00390 NEXT N
00400 RETURN
00410 REM ***** EFFECT 4 *****
00420 FOR N=1 TO 100 STEP INT(RND*5)+1
00430 POKE 1,20
00440 I=USR(0,256+N)
00450 NEXT N
00460 RETURN
00470 REM ***** EFFECT 5 *****
00480 FOR N=100 TO 1 STEP -INT(RND*5)-1
00490 POKE 1,20
00500 I=USR(0,256+N)
00510 NEXT N
00520 RETURN
00530 REM ***** EFFECT 6 *****
00540 FOR N=1 TO 30
00550 POKE 1,1
00560 I=USR(0,INT(RND*256)*256+1)
00570 NEXT N
00580 RETURN
00590 REM ***** EFFECT 7 *****
00600 T=INT(RND*100)
00610 FOR N=T TO 1 STEP -INT(RND*10)-1
00620 POKE 1,INT(RND*100)
00630 I=USR(0,(T-N+1)*256+T-N+1)
00640 NEXT N
00650 RETURN
```

Personally, I like the 'Effect 1' most of all. I think it's a real marvel and I'll eat my hat if you don't like it. If you want to enjoy it separately from the others, just type in:

```
270 GOTO 200
GOTO 200
```

Microbee Music

And Now the Theory

What makes the sound-effects so exciting and how can we produce them just by switching on and off the current through the speaker?

Every periodic physical phenomenon, such as sound, can be represented by the sum of the period frequency and its multiples. The period frequency is called the fundamental and the multiples are called harmonics.

The quality of a sound depends on the number of harmonics and their prominence. This is what we did in our sound-effect program: we varied the number of harmonics and their prominence and we did it at random or periodically with different periods, or even with the periods randomly changing. That's why we got such a variety, using only the simple on-off switching principle.

Type in Listing 1.

After starting it with RUN, you should:

1. See a picture of a square wave with a varying duty cycle (the duty cycle is the ratio of the ON interval to the sum of the ON and OFF intervals – that is, the period).
2. See a graphic representation of the relative prominence of the fundamental and harmonics up to the 20th harmonic.
3. Hear a sound corresponding to the waveform just being shown.

Unfortunately, you can't hear the sound when the graphics start moving – the reason has been already explained earlier in this article. The sound is always produced after the corresponding waveform has been displayed, together with its frequency-domain representation. (That's what the graph of harmonics is called, as opposed to the actual waveform shape, which is often referred to as the time-domain representation.)

Harmonic Analysis

This technique of analysing waveforms, which provides pictures about the harmonics, is called harmonic analysis. It is an indispensable scientific tool.

Watching the harmonics move, and listening to the sound, you'll notice how the shape of the waveform directly influences the quality of the sound you hear.

The total content of harmonics increases with the duty cycle decreasing from 50 per cent, while the fundamental is continuously diminishing. A pure 50 per cent duty-cycle square wave has only odd harmonics – or, rather, their 'envelope' in the spectrum often has a periodic character.

Does it matter if we swap the lengths of the ON and OF intervals? No, as far as sound is concerned. The only difference would be in the direct-current component (mean value) of the signal – the absolute values of both the fundamental and the harmonics remain unchanged and, as you can't hear the DC – the resulting effect is the same. (Moreover, there is a coupling capacitor in your MicroBee, protecting your speaker against an excessive DC bias.)

In practice, it means you don't have to worry which of the two intervals is actually 'speaker bit on' and which is

Listing 1.

```
00100 REM *** SQUARE WAVE HARMONIC ANALYSIS ***
00110 DATA 46,0,89,80,62,248,211,2,21,32,253,62,184,211,2,29,32
00120 DATA 253,45,32,237,201
00130 FOR I=0 TO 21
00140 READ D
00150 POKE I,D
00160 NEXT I
00170 CLS:LORES
00180 CURS 1,2
00190 PRINT "SQUARE WAVE HARMONIC ANALYSIS :  DUTY CYCLE"
00200 CURS 3,15
00210 PRINT "1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20";
00220 FOR T=63 TO 123 STEP 3
00230 GOSUB 290
00240 NEXT T
00250 FOR T=123 TO 63 STEP -3
00260 GOSUB 290
00270 NEXT T
00280 GOTO 220
00290 D0=(126-FLT(T))/63
00300 CURS 44,2:PRINT INT(D0*50);"% ";CURS 1,1
00310 PLOTR 0,47 TO T,47:PLOT T+1,47 TO 126,47
00320 PLOT 0,46 TO T,46:PLOTR T+1,46 TO 126,46
00330 A0=1/(1+D0)
00340 FOR N=1 TO 20
00350 A=INT(ABS(SIN(FLT(N)*3.14*A0)/FLT(N)/3.14)*100)
00360 K=N*6-2
00370 I=0
00380 IF A=0 THEN 420
00390 FOR I=1 TO A
00400 SET K,I+6
00410 NEXT I
00420 FOR J=I+7 TO 47
00430 IF POINT(K,J)=0 THEN NEXT: J 460
00440 RESET K,J
00450 NEXT J
00460 NEXT N
00470 POKE 1,1000
00480 I=USR(0,(63+T)*256+126-T)
00490 RETURN
```

Listing 2.

```
00100 DATA 46,0,89,80,62,248,211,2,21,32,253,62,184,211,2,29,32
00110 DATA 253,45,32,237,201
00120 FOR I=0 TO 21
00130 READ D
00140 POKE I,D
00150 NEXT I
00160 POKE 1,INT(RND*30)+1
00170 T=INT(RND*100)+1
00180 FOR N=0 TO 2*T STEP INT(RND*20)+1
00190 M=INT(ABS(FLT(N-T)))
00200 I=USR(0,(T+1-M)*256+M+1)
00210 NEXT N
00220 GOTO 160
```

Listing 3.

```
ADDR   CODE   LINE   LABEL  MNEM    OPERAND
0000          00100         ORG 0
0000 010000   00110         LD BC,0         ;'ON' INCREMENT DUMMY VALUE
0003 110000   00120         LD DE,0         ;SAME FOR 'OFF' INTERVAL
0006 D9       00130 LOOP0:  EXX             ;SWAP REGISTER BANKS
0007 3EF8     00140         LD A,248        ;LOAD 'SPEAKER BIT ON' VALUE
0009 2A2E00   00150         LD HL,(ONINT)   ;LOAD 'ON' INTERVAL LENGTH
000C 09       00160         ADD HL,BC       ;INCREMENT
000D 222E00   00170         LD (ONINT),HL   ;STORE AGAIN
0010 D302     00180         OUT (2),A       ;OUTPUT TO PORT B
0012 2B       00190 LOOP1:  DEC HL          ;DECREMENT 'ON' COUNT
0013 7D       00200         LD A,L          ;GET LSB OF 'ON' COUNT
0014 B4       00210         OR H            ;TEST THE COUNT FOR ZERO
0015 20FB     00220         JR NZ,LOOP1     ;LOOP FOR SPEAKER BIT ON
0017 3EB8     00230         LD A,184        ;LOAD 'SPEAKER BIT OFF' VALUE
0019 2A3000   00240         LD HL,(OFFINT)  ;LOAD 'OFF' INTERVAL LENGTH
001C 19       00250         ADD HL,DE       ;INCREMENT
001D 223000   00260         LD (OFFINT),HL  ;STORE AGAIN
0020 D302     00270         OUT (2),A       ;OUTPUT TO PORT B
0022 2B       00280 LOOP2:  DEC HL          ;DECREMENT 'OFF' COUNT
0023 7D       00290         LD A,L          ;GET LSB  OF 'OFF' COUNT
0024 B4       00300         OR H            ;TEST FOR ZERO
0025 20FB     00310         JR NZ,LOOP2     ;LOOP FOR SPEAKER BIT OFF
0027 D9       00320         EXX             ;SWAP REGISTER BANKS
0028 7D       00330         LD A,L          ;GET LSB OF TONE LENGTH COUNT
0029 B4       00340         OR H            ;TEST THE COUNT FOR ZERO
002A 20DA     00350         JR NZ,LOOP0     ;LOOP UNTIL ZERO
002C D9       00360         EXX             ;SWAP REGISTERS BACK
002D C9       00370         RET             ;RETURN TO BASIC
0002          00380 ONINT:  DS 2            ;'ON' INTERVAL INITIAL LENGTH
0002          00390 OFFINT: DS 2            ;'OFF' INTERVAL INITIAL LENGTH
0000          00400         END
00000 Total errors

LOOP2   0022    OFFINT  0030    LOOP1   0012    ONINT   002E
LOOP0   0006
```

'speaker bit off' – it simply doesn't matter.

Now, let's go back to our sound-effects. The program in Listing 2 is a somewhat modified version of the 'Effect 1' we produced before.

The lines 100-150 'POKE' our machine-language sub-routine into the memory.

The tone length is initialised on line 160 as a random number in the range of 1-30. (This isn't a mistake: the expression INT(RND*N) returns a random number in the interval 0,N-1 since RND is always less than 1.)

The variable T, assigned on line 170, is a half of the period of pulse-width modulation of the square-wave signal. This means we're going to periodically change the duty cycle of the waveform.

The step of the pulse-width modulation is a random number 1-20, as seen on line 180.

Not So Confusing

Line 190 looks rather confusing, thanks to MicroWorld BASIC, but isn't difficult to understand: the value of M changes from T to O and back to T again during one modulation period. (The modulation period is given by one complete FOR-NEXT loop between the lines 180-210 in our program.)

Finally, line 200 calls the machine-code sub-routine. Both ON and OFF interval lengths vary from 1 to T + 1, but in the opposite phase: their sum remains constant. This means the pitch remains constant during one complete FOR-NEXT loop. Only the quality of the sound changes.

This approach does have shortcomings. The modulation is done by BASIC and, therefore, the change of frequency or duty cycle can't be made smooth – you can actually hear the intervals between steps. This brings about an additional low-frequency modulation of the signal (which might be useful in many cases, because some of the resulting effects are really interesting).

But wouldn't it be nice to have a sub-routine, which could change all the tone parameters smoothly, producing a synthesiser-like gliding sound?

It would, and the sub-routine is in Listing 3.

The principle remains the same. Notice, however, this modification:

1. All parameters are now 16-bit values, rather than eight-bit – this extends the tone range.
2. Both the ON and OFF interval values are incremented once during each tone period.

The increment can be zero, in which case this sub-routine resembles the old one. It can, however, be a positive or negative number and we'll see in a while what can be done with it.

Type in the program in Listing 4.

Lines 100-170 load our new sub-routine to the memory. Lines 180-220 ask you about the tone parameters; the ON and OFF increments can be negative, positive or zero.

Passing The Parameters

Notice how the parameters are passed to the machine-code sub-routine: each is POKEd as a 16-bit quantity into two successive memory locations, the least significant byte first. The specific addresses for each tone parameter are obvious from the program.

Run the program and, as the program prompts, successively enter:

1000,1,-1,0,999

To have some more fun, try these combinations:

1000,1,-1,1,999
5000,1,-10,10,490
1,1,1,1,1000
1,1,1,0,1000

Listing 4.

```
00100 DATA 1,0,0,17,0,0,217,33,0,0,217,62,248,42,51,0,9,34,51,0
00110 DATA 211,2,43,125,180,32,251,62,184,42,53,0,25,34,53,0
00120 DATA 211,2,43,125,180,32,251,217,43,125,180,32,217,217
00130 DATA 201
00140 FOR I=0 TO 50
00150 READ D
00160 POKE I,D
00170 NEXT I
00180 INPUT "ON INTERVAL : ";N
00190 INPUT "OFF INTERVAL : ";F
00200 INPUT "ON INCREMENT : ";B
00210 INPUT "OFF INCREMENT : ";D
00220 INPUT "TONE LENGTH : ";L
00230 POKE 51,N:POKE 52,N/256
00240 POKE 53,F:POKE 54,F/256
00250 POKE 8,L:POKE 9,L/256
00260 POKE 1,B:POKE 2,B/256
00270 IF B)=0 THEN 290
00280 POKE 1,INT(FLT(B)+65536):POKE 2,INT((FLT(B)+65536)/256)
00290 POKE 4,D:POKE 5,D/256
00300 IF D)=0 THEN 320
00310 POKE 4,INT(FLT(D)+65536):POKE 5,INT((FLT(D)+65536)/256)
00320 I=USR(0)
00330 GOTO 180
```

Listing 5.

```
00100 REM *** ODE TO JOY ***
00110 DIM T(65),L(65),X(13)
00120 DATA 1,0,0,17,0,0,217,33,0,0,217,62,248,42,51,0,9,34,51,0
00130 DATA 211,2,43,125,180,32,251,62,184,42,53,0,25,34,53,0
00140 DATA 211,2,43,125,180,32,251,217,43,125,180,32,217,217
00150 DATA 201
00160 FOR I=0 TO 50
00170 READ D
00180 POKE I,D
00190 NEXT I
00200 DATA 20,20,21,23,23,21,20,18,16,16,16,18,20,20,18,18,0
00210 DATA 20,20,21,23,23,21,20,18,16,16,16,18,20,18,16,16,0
00220 DATA 18,18,20,16,18,20,21,20,16,18,20,21,20,18,16,18,11,0
00230 DATA 20,20,21,23,23,21,20,21,18,16,16,16,18,20,18,16,16
00240 FOR I=0 TO 65
00250 READ T(I)
00260 L(I)=3
00270 NEXT I
00280 L(13)=2:L(29)=2:L(37)=2:L(38)=2:L(42)=2:L(43)=2:L(55)=2
00290 L(56)=2:L(64)=2
00300 L(12)=4:L(14)=4:L(28)=4:L(30)=4:L(48)=4:L(54)=4:L(63)=4
00310 L(65)=4
00320 FOR I=0 TO 65
00330 PLAY T(I),L(I)
00340 NEXT I
00350 PLAY 0,6
00360 DATA 196,184,172,163,153,144,136,128,121,114,107,100,95
00370 FOR I=0 TO 12
00380 READ X(I)
00390 NEXT I
00400 FOR I=-1 TO 65
00410 IF I)-1 THEN 460
00420 N=800:F=800
00430 B=-4:D=-4
00440 W=142
00450 GOTO 540
00460 IF T(I)=0 THEN 660
00470 E=X(T(I)-11)
00480 B=4:D=-4
00490 N=2*E:F=N
00500 W=L(I)*4500/(F+N)
00510 IF I(65 THEN 540
00520 B=-1:D=1
00530 W=N-1
00540 POKE 51,N:POKE 52,N/256
00550 POKE 53,F:POKE 54,F/256
00560 POKE 8,W:POKE 9,W/256
00570 POKE 1,B:POKE 2,B/256
00580 IF B)=0 THEN 600
00590 POKE 1,INT(FLT(B)+65536):POKE 2,INT((FLT(B)+65536)/256)
00600 POKE 4,D:POKE 5,D/256
00610 IF D)=0 THEN 630
00620 POKE 4,INT(FLT(D)+65536):POKE 5,INT((FLT(D)+65536)/256)
00630 K=USR(0)
00640 NEXT I
00650 END
00660 PLAY 0,L(I)
00670 NEXT I
```

Try to experiment yourself. If you want to exit from the sub-routine prematurely, perhaps due to improper input tone parameters, you have to do it by pressing the reset button, because our sub-routine ignores the BREAK key. Just press the reset button, hold it down for a while, then release it. Run again.

Notice that:

1. If both increments are equal to zero, the tone pitch and quality remains constant.
2. If both increments are positive, the pitch will decrease.
3. The pitch will increase with both increments negative.
4. If the increments are of opposite polarity, the one with greater magnitude prevails.
5. If the increments are of opposite polarity with the same non-zero magnitude, the resulting sound seems to 'rotate', resembling the popular Leslie effect.

Something Strange Happens

Some of the parameters are inter-related. For example, if you enter the ON interval length equal to 100, 'increment'=-1 and 'tone length'=1000, something strange happens: as the tone length is actually the number of the tone periods, and the 'increment' is added to the interval length of each period, it's obvious that we get a negative interval length before the tone ends.

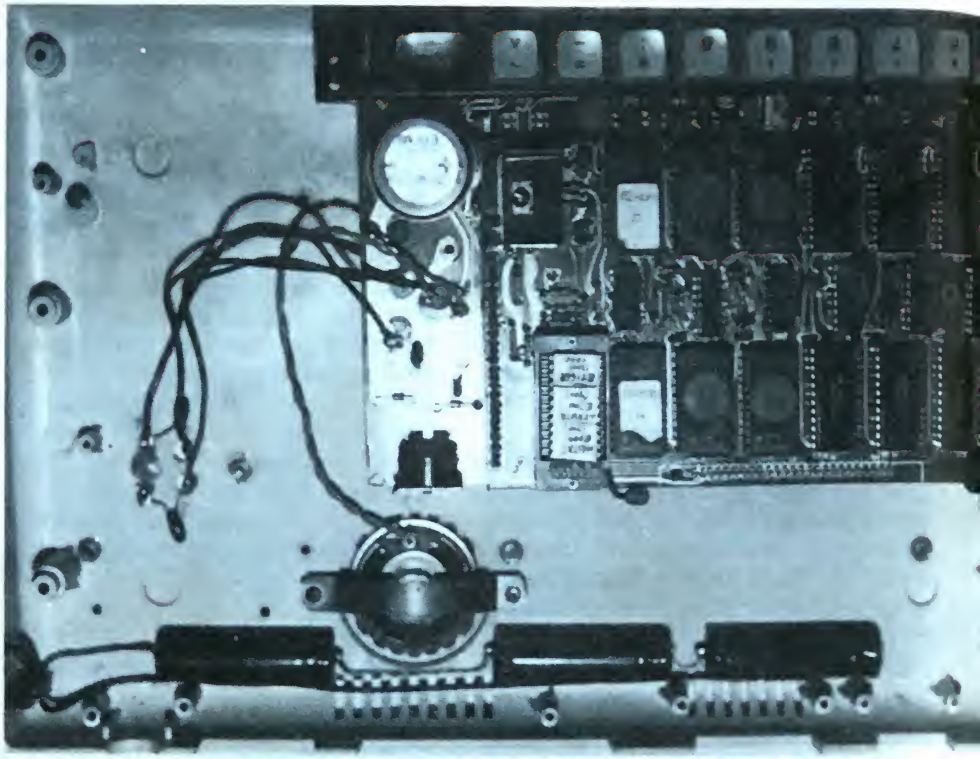
A negative 16-bit integer number N actually equals 65536-N, which means we suddenly get a very large number as the interval length. That's why you hear a very low-frequency clapping in such cases, and you have to use your reset button to bail out.

There is a simple formula to prevent this happening. If any of the increments is negative, it must fulfil this requirement: the absolute value of the increment times the tone length must be less than the initial interval. This is valid for both ON and OFF intervals.

The program in Listing 5, if typed correctly, should produce something resembling *Ode to Joy*. (Sorry, Ludwig!) The first part is produced using the standard MicroBee BASIC command PLAY. The second one is played by our new sub-routine. Hear the difference! ■

A Case of Prestige

Remounting the



Note that the three cells are mounted near the speaker and wired via the switch on the side of the case to the core board.

The red switch selects 2 MHz or 4 MHz operation.

Proud Bee kit builders who want to house their machine in one of the newer, more attractive, casings may encounter problems with incorporating the Bee's regulator. Eric Eulenstein did, and applied himself to overcoming the difficulty – here are the results of his ingenuity.

Kit Bee's Power Supply

FOR THOSE who, like me, were novices in the world of computing but were also adventurous and bought the original Bee in kit form, the greatest moment of all was when the final stage was complete and it worked (... or was it the sheer relief of finding no serious problems that meant calling in the experts?). From then on, learning to drive the beast (Beest?) and how to make it do as it was told has also been a lot of fun – the sheer power I have at my fingertips is amazing!

But time passed, and Applied Technology offered another case in which to house the Bee; and since it was much more attractive than the original flimsy shell, I could not resist getting one. Then emerged the problem of how to adapt the machine's single five-volt regulator to fit the new case.

It was mounted on the aluminium base plate which served as a heat sink for it. (Later models have two smaller units sharing the load, but with no heat sinks.) With the new case an alternative mounting had to be found, and rather than modify the circuit boards I decided to adapt the original regulator.

Using Existing Materials

Obviously a heat sink was needed, so I decided to use the original. I only needed to cut the aluminium base plate so it would fit neatly under the new case, in the area between the assembly screws at each end, and to cut a circular piece out at the speaker vent.

Then six 3 mm holes were drilled in the plate (four at the corners and two at the mid-point of the long sides), for securing it to the new base. By holding it in position and using it as a template, holes were drilled through the new base for securing with 3 mm bolts and nuts, and another two holes were drilled through the original regulator mounting holes. Then the positions of two 6 mm holes were carefully measured, marked and drilled for the two legs of the regulator to protrude inside the case.

Assembly was a matter of first placing the regulator in position on the underside of the new base, fitting the heat sink plate over it, then slipping two 3 mm bolts through the lot and screwing nuts on the inside (with a solder tag under one of the nuts). The regulator was now sandwiched between the case and the heat sink plate.

The six heat sink securing holes were then fitted with 3 mm nuts and bolts, but with a couple of washers between the plate and base to maintain the air gap set by the thickness of the regulator flange.

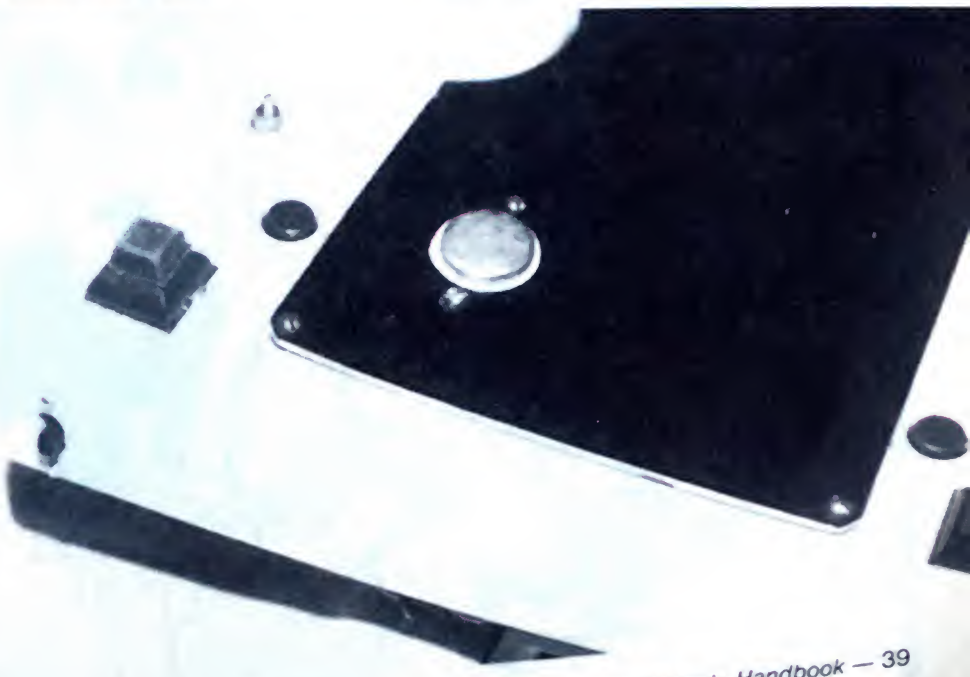
When fitted, the front edge of the circuit board under the keys was very close to the base because of its inclination, so the three bolts that came through needed to be as short as possible to prevent contact with the circuit board tracks or key tags.

Making the Connection

Finally, connection was made between the regulator and the main board via three lengths of stranded wire (with suitable colours such as red for 12 V in, orange for 5 V out, and green for earth) from the two pins of the regulator and the earth solder tag direct to the appropriate points on the main board. The speaker was then secured into its recess and also connected via another two similar lengths of stranded wire to the two points on the main board. The circuit board was secured into the new base at the appropriate points, and the top covers fitted.

One last task was to fit new rubber feet. Because the regulator protrudes further than the four feet on the base, new feet were required. I used four small, square, stick-on rubber feet on the original feet, but later applied four large feet beside the original ones with the smaller ones on top, to gain a little more space under the Bee for better air circulation.

Not only does my Bee now have greater eye appeal with the new case, and therefore is more prestigious, it does not have any hot spots on the case. What a mighty little machine! ■



Mounting the Bee on two sets of feet (one on top of the other) increases circulation of air under the machine and helps your Bee stay cool.

Microbee Variables

By Eric B Lindsay

Microbee owners don't yet have utility programs such as variable listers for programs, nor any method of locating variables in memory. These can be easily written if you understand precisely where and how the Microbee stores variables. This article explains how to locate and decode them.

MICROWORLD BASIC dedicates two bytes of memory as a pointer to the location in memory, of the value of each variable. It contains a variable table area from 1280 (500H) to 1696 (6A0H) or 416 bytes, in which are stored in strict alphanumeric order the 208 two-byte pointers for the real variables A0 to Z7.

There is another variable table area of 52 bytes from 1712 (6B0H) to 1764 (6B4H) for the pointers to the 26 integer variables A to Z, again in strict alphabetical order.

The pointers to any unused variable will contain 0. This makes it fairly easy to write a simple program in BASIC that goes through PEEKing at the variable table, and listing only those variables that are used. You can also PEEK locations from immediate mode.

You can use the monitor if you have Editor-Assembler or WordBee. However, as WordBee has a tendency to destroy BASIC programs, I would be cautious about using the monitor in WordBee to see what is stored in a program.

Each two bytes in the variable table print to the location of the start of the corresponding variable, and are in the usual low-byte, high-byte form. Those not familiar with hexadecimal arithmetic can find the location by multiplying the second (high) byte by 256, and adding the first (low) byte. This gives the location to PEEK for the start of the variable value.

The variables in a program are stored in the order in which they are encountered in the program, and – except for string variables – are stored immediately after the program in memory. BASIC takes care of where the program starts and ends, and where the next free space for numeric and string variables is, by means of another set of pointers which are located as follows:

Program start: 2256/7
Program end: 2258/9
Next available variable location: 2264/5
Next string variable location: 2268/9
Value to which SD is set: 2240

When you do a cold start, these pointers have values placed in them to ensure that BASIC programs start at 2304 (900H), and that the variable pointer is set to the next location in memory and the string pointer to 256 (100H) below the top of memory. Incidentally, you'll find these cold-start values stored in your BASIC 5.10 EPROMS, starting at BA3AH.

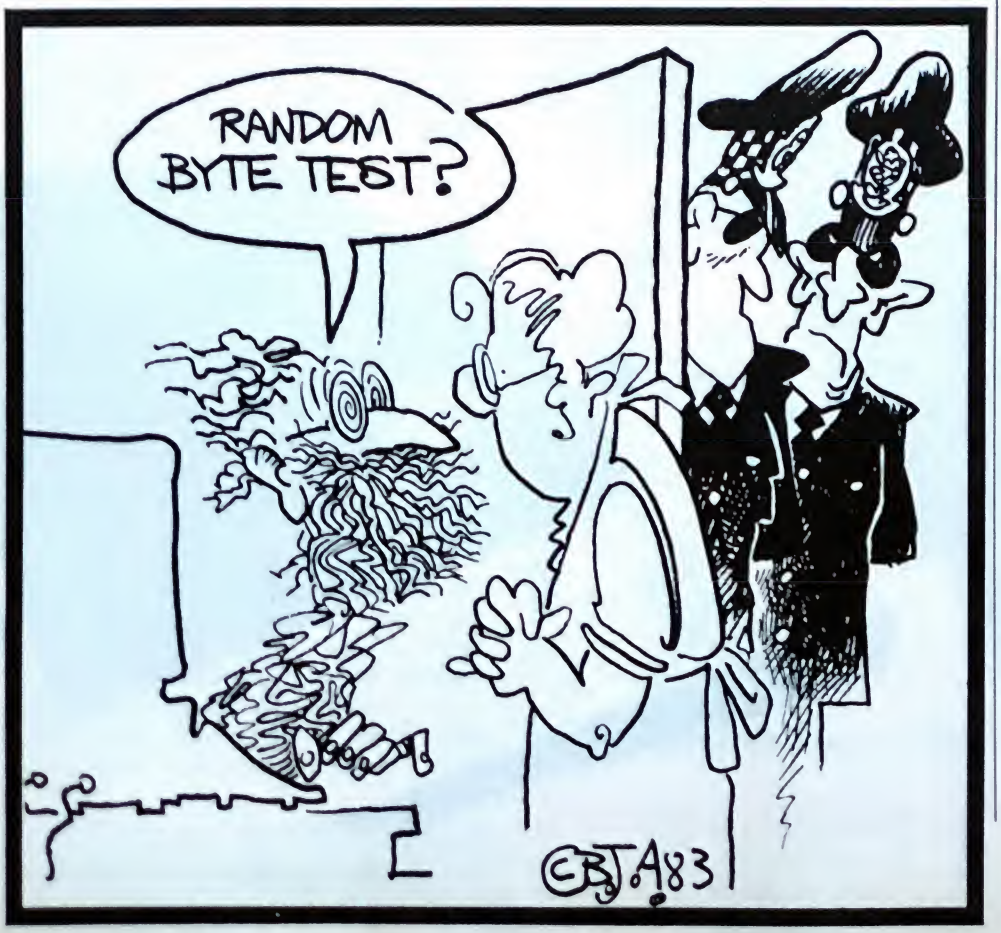
The values of the integer variables A to Z are stored as two bytes per variable, the same as pointers, in low-byte, high-byte form. You can find the value by multiplying the second byte by 256 and adding the first byte. You'll notice that the results of this give the value of the variable if it is between 0 and 32767. Negative numbers do not work so well.

In fact, at first glance, they don't appear to work at all. However, if you subtract the number you calculated from 65536, you'll come to the value of the variable. Negative numbers tend to be stored in computers in two's complement form, a complete description of which appears in most text books, or you can read Les Bell's article on binary and other arithmetics in the November and December 1981 issue of *Your Computer*.

Integer arrays are more complicated. When you dimension an array, say DIM A(2,3), you find that the array is actually three by four, since it starts from zero and goes to two, and from zero to three. Enough space is set aside in memory for each of the array variables stored two bytes each.

In memory, you'll find each array starting with 255, followed by the number of dimensions stored as a single byte. This is followed by a two-byte value for the size of each dimension except for the first one. The last two bytes are the size of the array in bytes. The actual values are stored after this, in the same form as regular integer variables.

Real variables A0 to Z7 are more interesting. The number of bytes used to store a real variable depends on the value you assigned to SD from BASIC.



You require half the SD value in bytes, plus one byte. Since the normal setting for SD is 8, this means each variable occupies five bytes. If SD is four, you need three bytes; if SD is 14, you need eight bytes per real variable.

If this bit is on (if the value of the byte exceeds 128 or 80H), then the number is positive. If lower, the number is negative. The next lowest bit (bit six) is set on for positive exponents, and off for negative exponents. The remaining six bits of the first byte are the value of the exponent of the number, which explains the allowable range of the MicroBee's real variables – approximately plus and minus 64. The bytes that follow contain the actual number, two digits per byte, in binary coded decimal (BCD) form. I must admit I have never encountered BCD before in a home computer.

BCD is really easy to understand, if you happen to speak in binary or hexadecimal. Each byte contains eight bits, and these are taken four at a time, and used to code the 10 decimal digits zero to nine. The surplus bit patterns, which in hexadecimal would be indicated as A to F, are simply not used. This means that if you view the location of real variables using a monitor, which displays in hexadecimal, you will actually see the value of the variable displayed. However, life wasn't meant to be easy, so if you view these bytes as decimal numbers, they make little sense. You have to convert them to binary, take the top four bits and the bottom four bits separately, and convert these back to decimal. I told you that you needed to read Les Bell's articles on binary arithmetic...

Arrays of real variables are treated much the same as integer arrays. The first byte stored is 255, followed by the number of dimensions (the maximum allowed is 255, but limits of line length will prevent you getting anywhere near that limit) then the actual dimensions are stored, two bytes per dimension, in low-byte, high-byte form (except for the first dimension, which doesn't appear to be stored). After these is a two-byte count of the number of memory locations the array occupies. The values in each array variable come next, and these are stored in exactly the same manner as normal real variables.

The real fun comes in decoding string variables. The pointer in the real-variables pointer area points to one variable-storage area above the program. Here, you find zero, followed by a two-byte pointer in low-byte, high-byte format, followed by a count of the number of characters in the string, followed by more zeros to bring the total memory used to the same total as an ordinary real variable.

If you follow the pointer, you'll find the first string variable is stored 256 bytes below the top of memory, and that it is stored in reverse – that is, if you have a string "abcd", it's stored as "dcba".

Working backwards from the location at which the string is stored, near the top of the memory, you have the ASCII codes for the string, followed by 128, which indicates the end of the string, followed by yet another pointer. This points to the pointer that pointed to the string! At the end (or bottom, if you like) of all the string variables, you will find a 255.

That covers all the variables, as stored in the MicroBee. The program listed will find and display all variable storage areas, and also the program area.

As programs tend to occupy a lot of space, you may prefer to use it without lines 410 to 470. The results of the program are shown, with it displaying variables as used within itself. Lines 100 to 200 can be changed in any way, and merely include a fairly wide range of variables to give the program something to do. All sections of the program will run independent of the others, provided you include line 270. ■

```
00100 REM DISPLAYS VARIABLES, TABLES AND PROGRAMS AS Nos 14/2/83
00110 LET A = 100
00120 LET B = -100
00130 LET C = A * B
00140 LET A0 = 100
00150 LET B7 = -100
00160 LET C0 = 1.2345678E08
00170 LET D7 = -1.2345678E07
00180 LET E0 = 9.8765432E-06
00190 LET F7 = -9.8765432E-05
00200 G0$ = "abcd"
00210 DIM G7(2,3)
00220 DIM H0(1,4)
00230 LET G7(0,0) = 4567.890
00240 LET G7(1,1) = 7654
00250 LET H0$(0,0) = "defg"
00260 LET H0$(1,4) = "xwvu"
00270 Z = 2256:J=0
00280 PRINT:PRINT "INTEGER VARIABLE POINTERS"
00290 FOR X = 1712 TO 1764
00300 IF J = 0 THEN PRINT: PRINT X;
00310 J = J + 1:IF J = 16 THEN LET J = 0
00320 PRINT PEEK(X);
00330 NEXT X
00340 PRINT:PRINT "REAL VARIABLES POINTERS"
00350 PRINT"Press any key when ready"
00360 IF KEY$ = ""THEN GOTO 360
00370 J=0:FOR X = 1280 TO 1696
00380 IF J = 0 THEN PRINT:PRINT X;
00390 J = J + 1:IF J = 16 THEN LET J = 0
00400  PRINT PEEK(X);: NEXT X
00410 PRINT:PRINT "PROGRAM AREA"
00420 J=0:PRINT  "Press any key when ready"
00430 IF KEY$ = "" THEN GOTO 430
00440 FOR X = PEEK(Z)+(256*PEEK(Z+1)) TO PEEK(Z+2)+(256*PEEK(Z+3))
00450 IF J = 0 THEN PRINT:PRINTX;
00460 J =J + 1:IF J = 16 THEN LET J = 0
00470 PRINT PEEK(X);:NEXT X
00480 PRINT:PRINT  "VARIABLES AREA"
00490 J=0:PRINT "Press any key when ready"
00500 IF KEY$ = "" THEN GOTO 500
00510 FOR X=PEEK(Z+2)+(256*PEEK(Z+3)) TO PEEK(Z+8)+(256*PEEK(Z+9))
00520 IF J = 0 THEN PRINT:PRINT X;
00530 J = J + 1:IF J = 16 THEN LET J = 0
00540 PRINT PEEK(X);: NEXT X
00550 PRINT:PRINT "STRING VARIABLES"
00560 J=0:PRINT "Press any key when ready"
00570 IF KEY$ = "" THEN GOTO 570
00580 FOR X = PEEK(Z+12)+(256*PEEK(Z+13)) TO 32512
00590 IF J = 0 THEN PRINT:PRINT X;
00600 J = J + 1:IF J = 16 THEN LET J = 0
00610 PRINT PEEK(X);
00620 NEXT X
00630 PRINT:PRINT "PROGRAM START"PEEK(Z)+(256*PEEK(Z+1))
00640 PRINT"PROGRAM END"PEEK(Z+2)+(256*PEEK(Z+3))
00650 PRINT"NEXT VARIABLE  LOCATION"PEEK(Z+8)+(256*PEEK(Z+9))
00660 PRINT"NEXT STRING VARIABLE LOCATION"PEEK(Z+12)+(256*PEEK(Z+13))
00670 PRINT"VALUE OF SD NOW SET TO "PEEK(2240)

INTEGER VARIABLE POINTERS

 1712 15 15 17 15 19 15 0 0 0 0 0 0 0 0 0 0
 1728 0 0 180 15 0 0 0 0 0 0 0 0 0 0 0 0
 1744 0 0 0 0 0 0 0 0 0 0 0 0 0 0 182 15
 1760 0 0 178 15 0
REAL VARIABLES POINTERS
Press any key when ready

 1280 21 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1296 0 0 0 0 0 0 0 0 0 0 0 0 0 0 26 15
 1312 31 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1328 0 0 0 0 0 0 0 0 0 0 0 0 0 0 36 15
 1344 41 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1360 0 0 0 0 0 0 0 0 0 0 0 0 0 0 46 15
 1376 51 15 0 0 0 0 0 0 0 0 0 0 0 0 0 56 15
 1392 122 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1408 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1696 0

PROGRAM AREA
Press any key when ready

 2304 0 100 59 32 161 32 68 73 83 80 76 65 89 83 32 86
 2320 65 82 73 65 66 76 69 83 44 32 84 65 66 76 69 83
 2336 32 65 78 68 32 80 82 79 71 82 65 77 83 32 65 83
 2352 32 78 111 115 32 49 52 47 50 47 56 51 13 0 110 12

VARIABLES AREA
Press any key when ready

 3854 255 100 0 156 255 240 216 195 16 0 0 0 67 16 0 0
 3870 0 201 18 52 86 120 72 18 52 86 120 187 152 118 84 50
 3886 60 152 118 84 50 0 127 0 4 0 255 2 0 4 0 60
 3902 196 69 103 137 0 192 0 0 0 0 192 0 0 0 0 192
 3918 0 0 0 0 192 0 0 0 0 196 118 84 0 0 192 0
 3934 0 0 0 192 0 0 0 0 0 192 0 0 0 0 192 0 0
 3950 0 0 192 0 0 0 0 192 0 0 0 0 255 2 0 5
 3966 0 50 0 126 249 4 0 192 0 0 0 0 192 0 0 0
 3982 0 192 0 0 0 0 192 0 0 0 0 192 0 0 0 0
 3998 192 0 0 0 0 192 0 0 0 0 192 0 0 0 0 0
 4014 126 242 4 0 208 8 7 0 182 15 32

STRING VARIABLES
Press any key when ready

 32491 255 15 173 128 117 118 119 120 15 128 128 103 102 101 100 15
 32507 51 128 100 99 98 97
PROGRAM START 2304
PROGRAM END 3854
NEXT VARIABLE  LOCATION 4024
NEXT STRING VARIABLE LOCATION 32491
VALUE OF SD NOW SET TO  8
```

PEEK, POKE and Screen

Some people find PEEKs and POKEs in BASIC complicated enough, and turn pale at even a mention of machine code. Tim Berry has written a simple introduction to these concepts – and screen RAM – using a Microbee.

RECENTLY I WAS talking to a friend about computing and I mentioned Machine Code ... to which he replied, 'Machine Code! I'm just learning about PEEK and POKE, and I don't even understand what you'd use them for.' Well, the answers are many, but two applications are:

1. Testing specific screen locations to see what is there.
2. Quick single-character printing on the screen.

These applications involve PEEKing and POKEing to the Screen RAM.

In Microworld BASIC the 'PEEK (int)' command gets a number from the memory location specified by the integer (int), whereas the command 'POKE intl,int2' stores the integer int2 in memory location intl.

A section of memory called the Screen RAM is dedicated to the screen (TV screen or monitor), and contains numbers that represent different characters, irrespective of whether they are from the Standard Character Set or your own characters (PCGs). The computer is constantly updating the screen from the numbers in the Screen RAM.

On the Microbee the Screen RAM begins at memory location 61440 and ends at memory location 62463. Each location corresponds to a character position on the screen, the screen itself being divided into 1024 such positions (64*16). The top left-hand corner corresponds to memory location 61440 and the numbering increases across the line until the end, then back down to the beginning of the next line. As there are 64 positions in each line the second line begins at memory location 61504 (61440+64). The bottom right-hand corner corresponds to memory location 62463.

To print a character in a specific screen position simply POKE the character number to the corresponding Screen RAM location. To find out what character exists at a specific screen position, PEEK to the correct memory location. Here is an example:

```
100 CLS
110 POKE 61440,65
120 CURS 2,1
130 PRINT PEEK(61440)
```

This short program POKEs an 'A' into the top left-hand corner of the screen. It then PEEKs that location and PRINTs the number next to it. For printing single characters this use of the POKE command is quicker than the PRINT command, and you may have noticed that the CURS command is not needed. Testing screen positions with the PEEK command saves both time and memory, as you don't have to set up and examine a huge two-dimensional array.

Moving Across The Screen

Simulating movement across the screen using POKE is fairly straightforward. You simply POKE the character onto the screen, pause, then clear that location and POKE the character into the next location, and so on. The following program may make it clearer:

```
100 P=61952
110 CLS
120 FOR I=1 TO 64
130 POKE P,65        :REM 'A'
140 FOR J=1 TO 50: NEXT J
    :REM Pause
150 POKE P,32     :REM Space
160 P=P+1
170 NEXT I
```

To move more than one character at a time you move them in sequence. The computer runs so quickly they appear to move together.

```
100 P=61454   : Q=62431   : R=61488
110 CLS
120 FOR I=1 TO 16
130 POKE P,65: POKE Q,66: POKE R,67
135 IF I=16 THEN 170
140 FOR J=1 TO 50: NEXT J
150 POKE P,32: POKE Q,32: POKE R,32
160 P=P+1            : Q=Q-64            : R=R-1
170 NEXT I
```

To list the Standard Character Set run the following program:

```
100 CLS
110 FOR I=0 TO 127
120 CURS I*8
130 PRINT I;
140 POKE 61445=I*8,I
150 NEXT I
160 CURS 0
```

The PCG characters start at 128 in the character set. Thus the equivalent of the three commands PCG: CURS 0: PRINT"A" is the one command POKE 61440,193.

You may have noticed that you don't have to be in PCG mode to POKE PCG characters; it is possible to do the same with the Block Graphics Characters. Whenever you execute a high-res or low-res command your Microbee loads the Block Graphics Characters into the PCG RAM. You can use the previous program to list the PCG or low-res character sets by typing in the following modifications:

```
130 PRINT I=128;
140 POKE 61445=I*8,
    I=128
```

RAM

Program Using PEEK and POKE

The following listing is a simple game I wrote for the Microbee. It uses PEEK and POKE to check for empty spaces and to print characters onto the screen. Although this program is fairly self-explanatory, you don't need to understand how it works in order to enjoy playing it.

The idea of the game is to move from the left-hand side of the screen, through the simple maze and out the space on the right-hand side without exceeding a fixed time limit or getting 'squished'! The four keys '[', ']', 'Q', and 'A' control the movement left, right, up and down respectively. ■

```
00085 REM          S Q U I S H
00087 REM         BY  TIM BERRY
00089 REM-------------------------------------
00090 CLS: CURS 27,15: PRINT"SQUISH"
00095 REM--PCG DATA-----------------------------
00097 REM                WALLS                (193)
00100 DATA  60,60,60,60,60,60,60,60
00110 DATA  60,60,60,60,60,60,60,60
00115 REM                PLAYER           (194,195)
00120 DATA  0,1,3,3,1,3,7,11
00130 DATA  11,3,2,2,2,2,6,0
00140 DATA  0,0,128,128,0,128,192,160
00150 DATA  160,128,128,128,128,128,192,0
00155 REM                SQUISHERS        (196,197)
00160 DATA  0,0,63,63,63,63,63,63
00170 DATA  63,63,63,63,63,63,0,0
00180 DATA  0,0,252,252,252,252,252,252
00190 DATA  252,252,252,252,252,252,0,0
00195 REM--SET UP PCGS--------------------------
00200 FOR I=64528 TO 64607
00210 READ D: POKE I,D
00220 NEXT I
00225 REM--MAP DATA-----------------------------
00227 REM  MAP 1
00230 DATA 12,16,8,1,3,10,6,4
00232 DATA 2,14,11,7,5,13,9,15
00235 REM  MAP 2
00240 DATA 13,16,8,1,10,6,4,2
00242 DATA 14,12,9,5,7,3,15,11
00245 REM  MAP 3
00250 DATA 2,16,11,1,6,14,3,9
00252 DATA 12,7,5,8,4,13,10,15
00255 REM--SELECT RANDOM MAP--------------------
00260 R=INT(RND*3)*10: RESTORE 230+R
00265 REM--DRAW MAP-----------------------------
00270 PCG: FOR I=1 TO 16
00275 REM  D POINTS TO GAP IN ROW I
00280 READ D: FOR J=1 TO 16
00285 REM  CHR(160) IS A SPACE IN PCG MODE
00287 REM  PRINT"AA" IS A WALL CHAR (193)
00290 IF D=J THEN PRINT CHR(160);CHR(160);
      ELSE PRINT"AA";
00300 IF I=16 AND J=16 THEN 320
00310 PRINT CHR(160);CHR(160);
00320 NEXT J: NEXT I
00330 NORMAL: CURS 0
00335 REM  DRAW EXIT
00340 POKE 61502,193: POKE 61503,193
00350 POKE 61630,193: POKE 61631,193
00355 REM--SET UP PLAYERS POSITION--------------
00360 P=61632: Q=P: T=0
00370 POKE P,194: POKE P+1,195
00375 REM--SET UP SQUISHERS---------------------
00380 DIM X(8),F(8): L=61566
00385 REM--START--------------------------------
00390 PLAY 12;17;24;22;17;12;22;20
00395 REM--PLAYERS MOVEMENT---------------------
00397 REM  CHECK FOR SQUISHER & TIME
00400 IF PEEK(P)=196 OR T=240 THEN 900
00405 REM  INCREMENT TIME COUNTER & GET
00407 REM  MOVEMENT CHAR FROM KEYBOARD
00410 T=T+1: A=ASC(KEY)
00415 REM  P=OLD POSITION
00417 REM  CALC NEW POSITION (Q)
00420 IF A=93 THEN LET Q=P+2: GOTO 440
      ELSE IF A=91 THEN LET Q=P-2: GOTO 440
00425 REM  POKE 258,0 = QUICK KEY REPEAT
00430 IF A=65 THEN POKE 258,0: Q=P+64
      ELSE IF A=81 THEN POKE 258,0: Q=P-64
      ELSE 600
00435 REM  CLEAR OLD POSITION
00440 POKE P,32: POKE P+1,32
00445 REM  CHECK IF NEW POS IS A SPACE
00450 C=PEEK(Q): IF C=32 THEN LET P=Q: GOTO 490
00455 REM  CHECK FOR LAST SQUARE
00460 IF P=61566 AND A=93 THEN 800
00465 REM  CHECK FOR SQUISHER
00470 IF C=196 THEN 900
00480 PLAY 1
00485 REM  POKE CHAR INTO NEW POSITION
00490 POKE P,194: POKE P+1,195
00595 REM--SQUISHERS----------------------------
00597 REM  X(I)=POS  F(I)=MOVEMENT FACTOR
00600 FOR I=1 TO 8
00605 REM  Z IS TEMP VALUE OF X(I)
00610 Z=X(I): IF Z<>0 THEN 650
00615 REM  IF Z=0 THEN CALC RND START POS
00617 REM  RND COLUMN MOVING UP
00620 N0=RND: IF N0<0.5 THEN LET F(I)=-64:
      Z=62402+INT(RND*15)*4: GOTO 680
00625 REM  RND COLUMN MOVING DOWN
00630 IF N0<0.95 THEN LET F(I)=64:
      Z=61442+INT(RND*15)*4: GOTO 680
00635 REM  LAST COLUMN MOVING DOWN
00640 F(I)=64: Z=61498: GOTO 680
00645 REM  CLEAR OLD POSITION
00650 POKE Z,32: POKE Z+1,32
00655 REM  CALC NEW POS
00660 Z=Z+F(I): C=PEEK(Z)
00665 REM  IF NEW POS IS NOT A SPACE OR
00667 REM  THE PLAYERS CHAR, THEN X(I)=0
00670 IF C=32 OR C=194 THEN 680
      ELSE LET X(I)=0: GOTO 700
00675 REM  POKE CHAR INTO NEW POSITION
00680 POKE Z,196: POKE Z+1,197
00685 REM  SET X(I) TO NEW VALUE
00690 X(I)=Z
00700 NEXT I: GOTO 400
00795 REM--PLAYER WINS--------------------------
00800 PLAY 12;17 : FOR I=1 TO 2
00810 PLAY 24;22;12;7;19;17;17;12;22;20
00820 PLAY 12;7;19;17;12;17;24;22;12;7
00830 PLAY 19;17;17;12;24;17;22;12
00840 IF I=1 THEN PLAY 20;17;19;12
00850 NEXT I
00860 CURS 27,8: PRINT"WELL**DONE"
00870 CURS 27,9: PRINT" YOU**WIN "
00880 PLAY 17,8: GOTO 999
00895 REM--PLAYERS LOSES------------------------
00900 PLAY 1,2,8,2,12,2,10,2,8,2,1,2
00910 CURS 27,8
00920 IF T=240 THEN PRINT"TIME'S  UP"
      ELSE PRINT">SQUISHED<"
00930 CURS 27,9: PRINT" YOU LOSE "
00940 PLAY 1,8
00945 REM---------- ----------------------------
00999 CURS 0: END
```

Bee-Thoven

Many people support Australia's own Microbee, covering a range of activities from business to games. One of the many applications developed for this machine is the BeeThoven/BeeComposer hardware and software packages. Mike Newnham reviewed them recently.

NO DOUBT all users of the Microbee are familiar with the capabilities of its tone generator. It produces a series of beeps and pops that resemble tunes and, using machine code routines, it can be made to generate quite unusual and effective sound effects.

Milan Hudecek of the Melbourne-based firm Robotron, who wrote the article 'Microbee Music' which appears elsewhere in this magazine, is now marketing a piece of hardware called 'BeeThoven' which plugs into the MicroBee parallel port and takes sound synthesis on the Bee a few steps further.

Also available from Robotron is a program called 'BeeComposer', which is a machine code music composition program for use with BeeThoven.

The main elements of BeeThoven are three tone generators, one noise source and three noise/tone mixers. The frequency of each tone generator is independently programmable over a wide range. Noise may be mixed with each of the tone sources, and a programmable envelope generator is provided for amplitude modulation of each voice for functions such as attack and decay and programmable volume control.

All these features combine to permit the creation of a wide range of sound effects and musical notes. BeeThoven is supplied with a demonstration tape giving some idea of its capabilities, and after using the package for a time, I believe there are not many sounds it can't reproduce.

A software driver which allows your programs to control BeeThoven is also provided on the demo tape. All commands to control sound generation are passed to BeeThoven via this driver.

There are also two DB-9 plugs mounted in the case. These permit the direct attachment of either Atari or VIC-20 joysticks. The BeeThoven driver is used in this instance to read the ports and return the joystick position to the calling program. I found that running under BASIC the response is a little slow. This is no fault on the part of BeeThoven, but is due to the impracticality of using BASIC for real-time applications. Under machine code the performance was excellent, the only controlling factor being the efficiency of program coding.

The documentation supplied with BeeThoven is first class. It is presented in a spiral-bound booklet which leaves nothing out; each command is fully explained as to its function and use. There is a program on the demonstration tape called 'Tutor', which is used in conjunction with the manual to guide the user through the commands and explore the operation of each. The manual contains a table which specifies the data values required to generate each note of the tempered musical scale, and there are both BASIC and Assembler source listings of routines to generate various sounds.

While BeeThoven is not physically large, its size belies its abilities. It is a most impressive device. BeeThoven, demonstration tape and manuals together sell for $89.50.

Very Impressive Composer

For an extra investment of $29.50 you can purchase BeeComposer, which transforms 'impressive' into 'very impressive'.

BeeComposer is a machine code program which enables the user to create, transcribe and edit musical compositions, and immediately play them on BeeThoven. It supports all standard musical notation and provides for the selection of key signature, time signature and tempo. You can compose in three parts, or voices, and save all your compositions to tape for later playing or editing.

After loading, BeeComposer signs on with its own fanfare, then displays the great staff and below it the lower bass staff. Music is entered onto the staves by means of simple keystrokes. Cursor control makes the tasks of composition and editing very easy indeed. BeeComposer makes very good use of the MicroBee's high-resolution graphics and all characters are clearly recognisable as the musical elements they are designed to represent.

On the 'flipside' of the BeeComposer cassette, a demonstration composition is provided. This is loaded in via BeeComposer and, when played, is a perfect example of the magic of the BeeThoven-BeeComposer symbiosis.

One observation made by a number of people who have seen and heard this package in operation is that it has real potential as a means of teaching music. The user is able to write a piece and play it immediately. He or she may then edit the composition, replay it, and demonstrate instantly the behaviour of each individual element of a piece of music. Some students of music may find that the replacement of pencil and music book with keyboard and screen is a very enjoyable change.

One thing I discovered is that experimenting with BeeThoven and BeeComposer becomes almost as addictive as programming itself. Programming sessions that last until 5 am have revealed some very talented home programmers. Perhaps this offering from Robotron will awaken latent musical talents in some of them too.

Again, the manual supplied with BeeComposer is first class in both quality and presentation. It clearly explains the operation of every command used by BeeComposer, and provides a description of the use and function of each component of music notation.

To sum up, BeeThoven is certainly a worthwhile addition to your system. If you intend purchasing it, then I would recommend that you also buy BeeComposer, especially if you have a musical bent.

If Robotron maintains this standard of quality, we should see some excellent products from the company in the future. ■

Product Review Summary

Product:	BeeThoven	BeeComposer	Combined Package
Ratings:			
Ease of use:	Very good	Excellent	Excellent
Entertainment:	Good		Excellent
Educational:	Fair		Excellent
Documentation:	Excellent	Excellent	
Use of graphics:		Very good	
Value for money:	Very good	Excellent	Excellent
Holds interest:	Yes		Addictive
Price:	$89.50	$29.50	$119.00
Review unit from:	Robotron Pty Ltd, PO Box 232, Mooroolbark 3138.		

A problem which must plague most personal computer owners who use a tape recorder for cassette data storage is setting the volume control when loading a program. Eric Eulenstein built a simple monitor to help him get it right, and made some surprising discoveries in the process.

Volume Control of Program Loading

MOST COMPUTERS require a volume setting within a very small range before they will accurately load program material, and because of the variation in depth of recording on a tape with different recorder brands, and the varying playback responses of different brands of tapes, many microcomputer owners probably have a frustrating time establishing a suitable volume control setting for each tape.

Although originally used with a Microbee, the unit described here can be used with any personal computer and will simplify loading procedures. It also provides an audible signal which will help users develop efficient loading techniques, and even identify system troubles that would not otherwise be evident.

The Microbee has a 'clipping' circuit which automatically controls the input signal to the level required by the machine: providing the input signal is above a certain value, the loading of a program is assured. I bought a new tape recorder to use with my Bee and found reliable loading required the volume control to be almost on full. This did not appear to be out of the ordinary at the time, and performance was satisfactory at 300 and 1200 baud rates.

Eventually it dawned on me that a monitor speaker would be an asset, to determine the start of a program on a tape and so help with correct loading. I built one using a small speaker from an old transistor radio, but with a 500 ohm pot in series to regulate the sound produced by it. This mechanism became an important aid in the loading procedure, allowing me to quickly find the gap between programs; to cue up the start of a program and avoid false starts; to monitor the program and identify tape drop-outs; to check that the head azimuth was optimised; and so on.

Adjusting Your Azimuth

I'll diverge from my original course to explain that last-mentioned benefit. Have you ever tried to load somebody else's program tape, especially at 1200 baud, and received a BAD LOAD message? One of the reasons can be incompatibility between the azimuth of the tape recorder head and that of the data on the tape.

Most recorders allow access to an adjusting screw, usually via a small hole, allowing the azimuth to be set to its correct position. By using a suitable screwdriver and listening to the program while the recorder is playing, the recorder head can be adjusted for a particular tape by tilting it one way or the other with the adjusting screw (usually only a very small change is required) until the sound is as sharp or harsh as it can be made. This should then allow loading of the troublesome tape.

Don't forget you'll have to readjust the azimuth to suit one of your own tapes again, prior to SAVEing the new program onto a spare cassette! Then you will have a copy to suit your own equipment.

Back to the Story

After constructing my monitor speaker I decided to attach a meter to it, in order to be able to determine a minimum setting of the volume control for reliable loading. The first observation I made was that the meter was reading a strong signal *before* the program signal started. I was even getting a signal when there was no cassette in the recorder, but with the play key depressed. So, what was it all about?

Pulling out the AUX plug stopped this false signal and further checks showed that with the AUX plug removed I was able to load a program at a much lower volume control setting! Obviously an oscillatory action was created by the presence of a loop completed when the two plugs were inserted at the same time.

I dismantled the recorder and took a good look at the circuit board, which revealed that the two sockets were both earthed at different ends of the circuit board's earth track. Apparently the earth was reactive in some way, with each end connected separately into the com- ▶

puter, and stimulated by the operation of the play key.

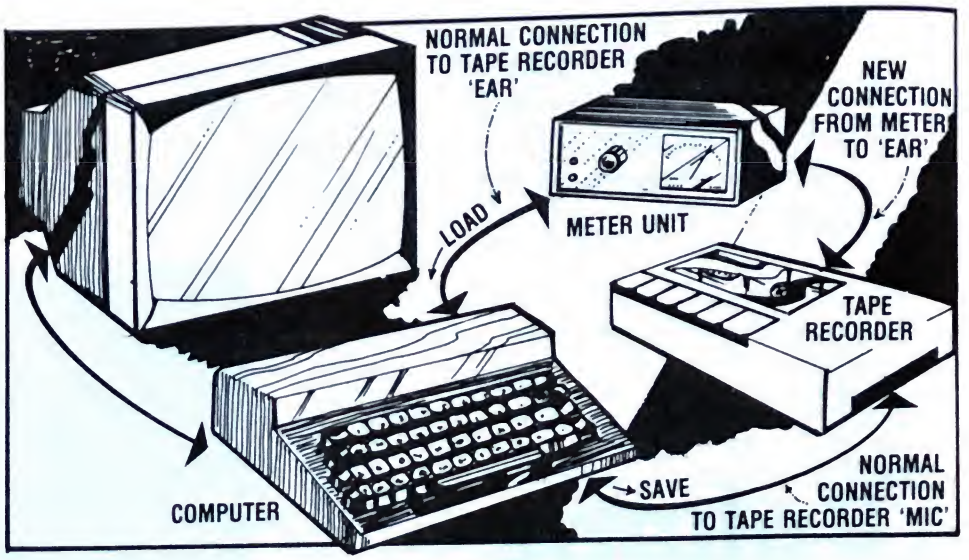
The recorder had been intended only for recording *or* playing, in an entirely different environment. But when using it with a computer I'd found it convenient to leave both plugs in permanently, and in this case the earth loop inside the machine became reactive and caused a super audio oscillation.

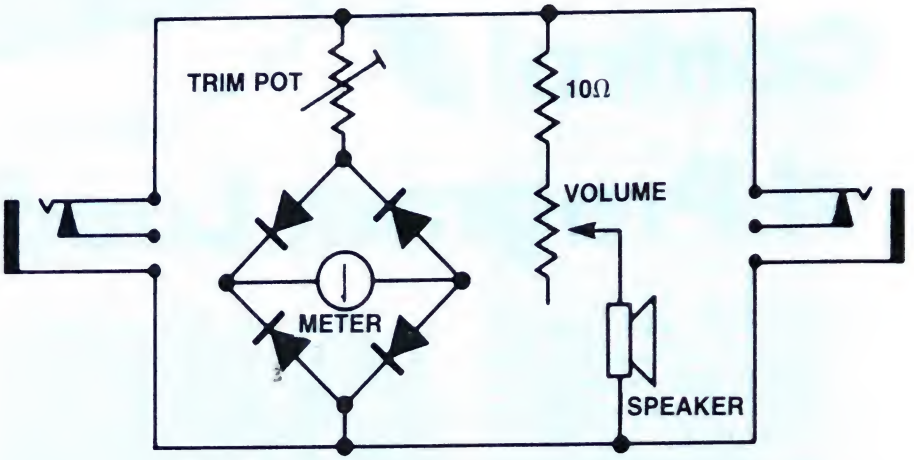
There is only one way to eliminate inductive reactance in a case like this: short circuit the inductance. So, I bridged the earth points at each jack, and (using a fine single-strand insulated wire) I connected them radially to various points in the earth loop on the circuit board, particularly to those points that are extensions to the centre of the board, to pick up the earthy end of bypass capacitors and so on.

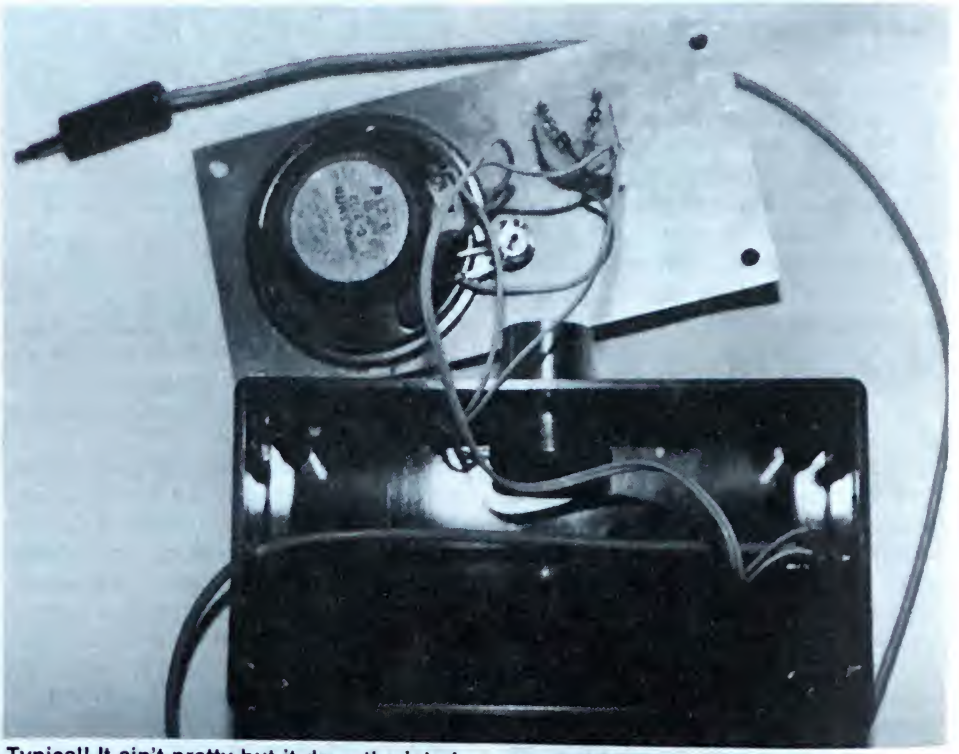
Success

On testing the reassembled machine I found the meter registered nothing other than program signal – great! I then set the volume control low enough to cause BAD LOADs on the MicroBee and adjusted a trim pot in series with the meter so it registered 2 on the scale of 10 – now it can always be used to set the volume to a measured level. (I usually aim at between 6 and 8 on the meter.)

For those who have computers which require the volume to be set at a special level, the meter can be used to establish the upper and lower limits. Then, by adjusting the volume to the mid-point, the meter reading can be set with the trim pot to, say, 8 on the meter scale. Thus, every time a program is to be loaded, by setting the volume control on the tape recorder to get a reading of 8 on the meter a reliable load is assured! ■







Typical! It ain't pretty but it does the job. I mounted all the bits in a cheap 'jiffy' box with the connectors on one end.

PARTS LIST

Jiffy box	(Dick Smith Cat. No. H2753)
Level meter	(Dick Smith Cat. No. Q2100)
Signal diodes	4 x OA91 or similar
Trim pot	5KΩ
Potentiometer	200Ω
Resistor	10Ω
Miniature sockets	2 x 3.5 mm
Stick-on rubber feet	4 x small

microbee engineering note book

Ideas, additions, improvements and service checklist for your microbee

This section contains ideas, additions, improvements and changes that can be made to your MicroBee. Many improvements are already in the latest MicroBees.

This is not the Technical Manual and does not replace it. Most of its ideas have been compiled by Max Maughan, Engineering Manager, Applied Technology.

These ideas and additions are for your interest and can be added to your MicroBee by you or someone with hardware knowledge. The only things that will be added by Applied Technology are the modifications required with upgrade. These will only be done with the upgrade and not under any other circumstances.

CONTENTS

FIELD CHECK LIST

Please note: This list is written for people with some electrical knowledge. You must know how to use a multimeter and a low-wattage soldering iron. In many cases a working MicroBee would be handy to be able to swap parts; when swapping parts put the suspect part into the working unit. For example, swap the 6545 from the dead unit to the working unit; this way if there is anything in the dead unit causing the chip to fail it won't damage the good one.

DISCONNECT BATTERY WHEN CHECKING UNIT

FAULT	CHECK	REMEDY
No power on or picture dead unit	Try another plug pack unit; if machine OK check wiring and voltages — should be no load 14-16 volts	Replace or repair plug pack
Ditto	Still dead, plug pack OK — remove cover, check input voltages at D14, check D14 for open circuit and D15 for short circuit	Replace D14; remove D15; don't replace D15
Ditto	Still dead — check all tag tantalum capacitors for burnt tops; with suspicious ones, remove one leg from board and power up	Replace capacitor
Ditto	Still dead — if input volts OK, check the three 5 volts: 1. pin 16 IC2, main board; 2. pin 16 IC3, main board; 3. pin 24 IC30, core board	Any 5 volts missing, check and replace 7805 reg.
Ditto	Still dead, all 5 volts OK — try another core board	Change ROMs, service core board
Ditto	Still dead, core board OK — change IC25 Z80CPU	Replace Z80
Ditto	Still dead, CPU OK — remove IC9 6545, leave out, and try for power-on beep	Replace 6545 if beep heard
Ditto	Still dead, 6545 out — change IC1 PIO, try for power-on beep	Replace IC1 PIO
Ditto	Still dead, all above OK — time for an experienced technician with test equipment	Take unit to MicroBee Service Centre
Power-on beep but no picture	Check leads from DIN plug to monitor for open circuits	Repair connections
Ditto	Still no video, leads OK — try running a BASIC program using play; test control 'G'	If no go then replace 6545
Ditto	Still no video, core board and 6545 OK — time for Service Centre	Service Centre
Picture but no beep	Check wiring to speaker and speaker for open circuit	Repair wiring
Ditto	Change IC1 PIO and check TR3 under keyboard	Replace PIO or TR3
Loss of picture after a couple of minutes	Check regulators under keyboard for loose screws stopping heatsinking	Tighten nuts or use pop rivets
Ditto	Regulators heatsinking OK — change IC9 6545	Replace 6545
Ditto	IC9 OK — try another plug pack	Replace or repair plug pack
Breaking out of programs back to ready	Change plug pack — bridge rectifier breaking down in plug pack	Replace bridge rectifier in plug pack
Only cursor on screen after 5 to 10 minutes	Change plug pack - same as above	Replace bridge rectifier in plug pack
Black bands moving up screen	Check DIN plug — move to see if bands change; if no change, try another plug pack	Replace plug pack
Keys not working	Resolder contacts while pressing hard on keytop; if still no go, remove and replace (use a solder sucker to remove solder from contacts and PC holes)	Resolder or replace
No keyboard input, incorrect video	Change IC9 6545	Replace IC9 6545

Know Your Limits When Attempting Repairs

The faults and remedies listed are the most common faults and easy to fix. Other less common faults are harder to locate. These faults involve removal of soldered-in ICs. Experience is required to locate correct ICs, and boards damaged by removal of chips will not be replaced. Normal *labour* costs will be charged to repair the damage. If in doubt use your nearest **MicroBee Service Centre.**

A good example of what can happen when you try to do something you don't know enough about happened in June 1983. An electronic magazine published an article on upgrading your MicroBee from 16K to 32K yourself at a fraction of the cost. Applied Technology's price for this upgrade was $125.00. A customer reading this article decided to do it himself.

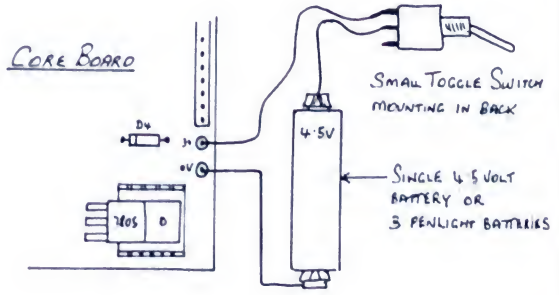
However, after his attempt to do the upgrade, the MicroBee didn't work any more, so Applied Technology's service department received the unit to restore it to working condition. Somehow this customer had managed to destroy all the RAM chips on the core board. To restore his MicroBee to a 16K unit cost him $130.00; five dollars less and he could have had a 32K instead of his original 16K.

Unless you have plenty of money to waste, know your limits when attempting repairs or connecting things to your MicroBee. If you're not sure how to do it, ask for advice from someone with the required knowledge.

BATTERY BACK-UP

To get longer life from your battery, switch it off when it is not needed. Even though battery back-up is great, it is still not as good and reliable as a good copy on tape. You never know when the battery will go flat.

If you must have battery back-up 24 hours a day, seven days a week, then install a rechargeable battery. By switching the battery off when it is not required I have had the same battery for over twelve months.



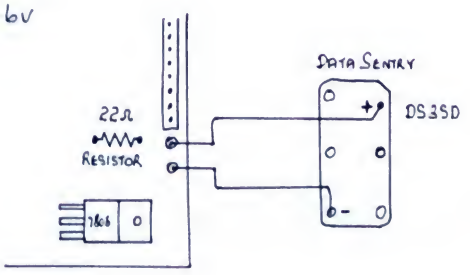
RECHARGEABLE

NICKEL - CADMIUM BATTERIES

GENERAL ELECTRIC DATA SENTRY

MODEL NO DS35D

VOLTAGE 3·6V



NOTE: These batteries are not stocked or sold by Applied Technology.

Facts About Plug Packs

These units have been tested and approved by their manufacturers to run the MicroBee. We have no control over the quality of parts used in these units.

It is normal for them to get warm after a period of time running under full load. The MicroBee uses between 900 ma and 1 amp and the plug packs are designed to supply 1 amp.

In the plug pack there are three components: 1. Transformer, 2. Bridge rectifier, 3. 2000 uf capacitor. The part that usually fails is the bridge rectifier. It is not the continuous supplying of 1 amp of current that causes this to fail, but the power-on surge current, which can be as high as 30 amps.

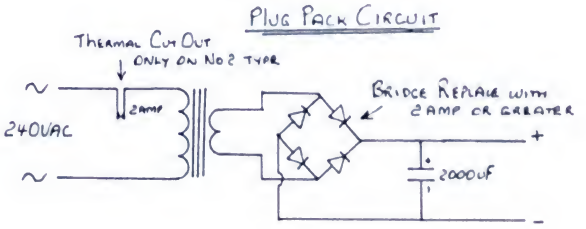
Under warranty the whole plug pack is replaced free of charge; this does not include the leads. Outside warranty customers are up for the cost of a new plug pack, but in most cases the plug pack can be repaired.

Repairing A Plug Pack

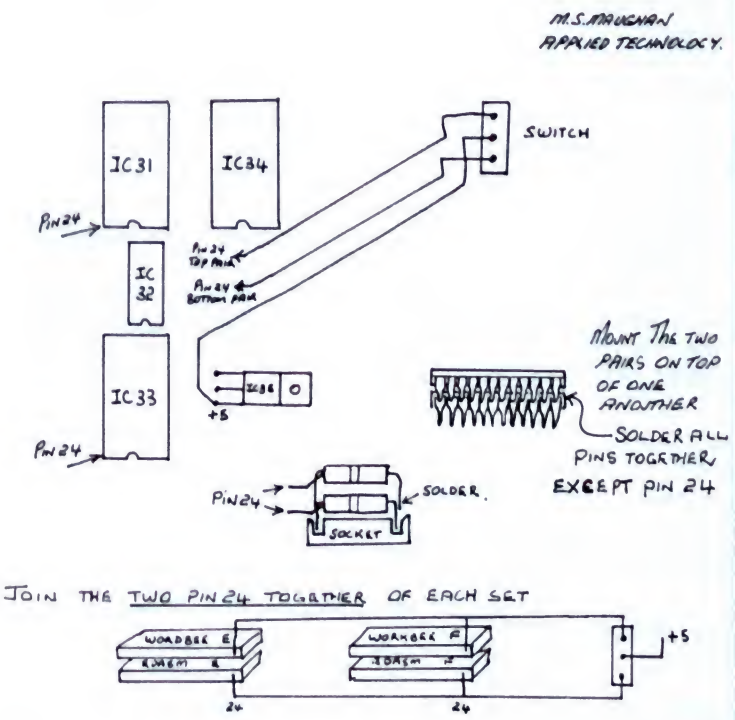
There are two types of plug packs: 1. The Arlec that plugs into the power point; 2. The MicroBee one, fitted on a lead. To repair no. 1, lay pack on its side on a hard surface and hit with a hammer along the joint, then pull apart. For no.2 type, unscrew four screws, then tap around the joint to get apart. Replace the bridge rectifier with another bridge or four diodes, then re-glue or screw back the cases.

Just as a point of interest, MicroBees don't mind being left switched on. For example, on August 24 1982 at the Gosford factory we wanted to find out the MTBF on the MicroBee. A MicroBee was switched on with a program running in a continuous loop, and was left on for 24 hours a day, 7 days a week. On July 14 1983 that MicroBee was still running, and not one component had failed. So when you're not using your MicroBee, put it to work; for example, use it as burglar alarm.

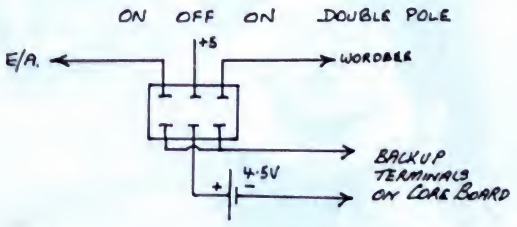
No failures in eleven months = 325 days = 7800 hours. This would be equivalent to using your MicroBee for three hours a day, 7 pm to 10 pm, seven days a week for 2600 days – which amounts to 7.12 years.



WORDBEE AND EDITOR ASSEMBLER IN THE SAME MICROBEE

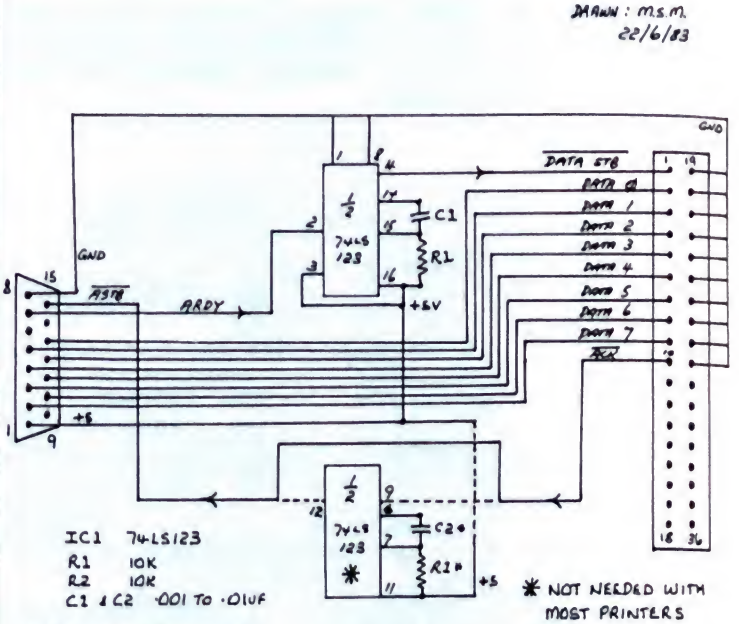


USING ONE SWITCH FOR ROM AND BATTERY BACK-UP



MICROBEE PRINTER INTERFACES

PARALLEL PRINTER

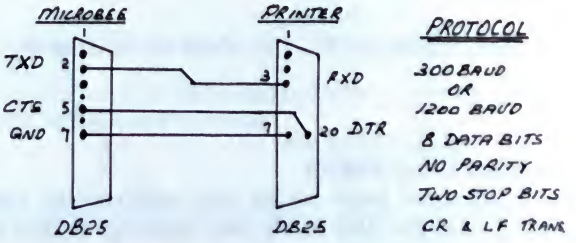


DB15 CONNECTIONS

1 — +5V	6 —	11 — DA4
2 — DA7	7 — ARDY	12 — DA2
3 — DA5	8 — GND	13 — DA0
4 — DA3	9 —	14 —
5 — DA1	10 — DA6	15 — ASTB

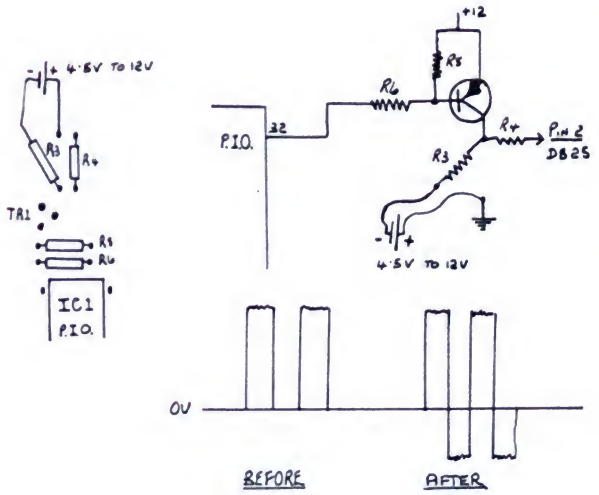
NOTE READY MADE CABLES SOLD BY APPLIED TECHNOLOGY.

SERIAL PRINTER RS232



IMPROVING RS232 OUTPUT ON THE MICROBEE

BY MAX MAUGHAN
APPLIED TECHNOLOGY.



The RS232 output from the MicroBee will not drive all RS232 receiving devices because it has no negative drive signal. If your receiver is a 1489 IC receiver you will have no problems. If the receiver has an input resistor you will probably have problems.

If you know the receiver circuit works because it worked on another machine with an RS232 transmitter circuit, OK. Then you need to try the above circuit modifications in your MicroBee. For a quick and easy test use a battery between 4 and 12 volts. Current drain from the battery is low so it could become permanent. Disconnect RS232 plug when not in use to save battery life.

Another suggestion is if your printer has a –12 volt supply, connect it to a spare pin on the DB25 plug, feed it back to the MicroBee and connect it to R3. The only other way – a rather untidy way – is a separate 12 volt supply.

REMOVING THE LINEFEED CHARACTER FROM THE LIST OUTPUT STREAM

Some printers, usually parallel interface models, perform both a carriage return and line-feed when they receive the ODH character (return). Since the MicroBee sends an OAH (line-feed) code in the listing as well as a return, double spacing results, wasting paper.

To remove this problem, a special routine can be placed into the spare RAM which intercepts the LF code before it gets to the output routine.

The assembly code to accomplish this is as follows:

```
0148    FE0A        cp      1F      ;is it a linefeed
014A    C8          ret     z       ;if it is,dont print it
014B    C3xxxx      jp      routine ;print the character
```

The address "xxxx" depends upon which stream is being used for the printer.

```
Parallel printer    address B5F8    xxxx = F8B5
300 Baud RS232      address A87A    xxxx = 7AA8
1200 Baud RS232     address A87F    xxxx = 7FA8
```

The 16-bit printer vector must also be changed to print to the intercepting routine:

```
Store 48H, 01H at addresses
00B4H, 00B5H for parallel printer (device 1)
00BAH, 00BBH for 300 baud RS232 (devise 4)
00BCH, 00BDH for 1200 baud RS232 (device 5)
```

In BASIC

To make the above changes from BASIC, run the following program after the MicroBee has been **cold started**

```
Parallel printer (device 1)
00100 DATA 254,10,200,195,248,181
00110 RESTORE 100
00120 FOR I=328 TO 333: READ A : POKEI,A: NEXT I
00130 POKE 180,72: POKE 181,1

300 Baud Serial Printer (device 4)
00100 DATA 254,10,200,195,122,168
00110 RESTORE 100
00120 FOR I=328 TO 333: READ A : POKEI,A: NEXT I
00130 POKE 186,72: POKE 187,1

1200 Baud Serial Printer (device 5)
00100 DATA 254,10,200,195,127,168
00110 RESTORE 100
00120 FOR I=328 TO 333: READ A : POKEI,A: NEXT I
00130 POKE 188,72: POKE 189,1
```

IMPROVING KEYSWITCH RELIABILITY

To improve the reliability of the keyswitches on the MicroBee, a resistor network, RN1, has to be changed.

This resistor network is in front of IC4 under the keyboard frame. It is used as a pull-up resistor on the input lines of IC4 and does not appear on the circuit diagram.

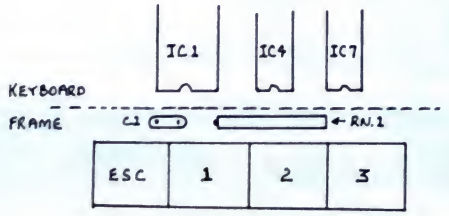
The resistance value of this network is 3k3 or 4k7, which is too low. The resistance has to be increased to 33k (or near to this value); a 33k network or eight 33k resistors will do.

To remove the old network, pull out keyswitches nos. 1 and 2; remove them as described in the notes on repairing keyswitches. With switches removed, suck out solder from the resistor network connections. Remove resistor through keyswitch holes.

Fit new resistor pack, then refit keyswitches. If you have to use resistors they may not fit under the keyboard frame. If you cannot fit them, solder them on the bottom; there is plenty of room underneath between the base and the case.

If you have a machine with faulty keyswitches it is advisable to change the resistor network first. Most if not all of your faulty keys will disappear.

This repair can be done by the service department, but the normal repair charges will apply if the machine is out of warranty. It is not necessary in most cases to change this resistor if key operation is okay.



REPAIRING KEYSWITCHES ON THE MICROBEE

Tools required: 1. soldering iron; 2. solder sucker; 3. long-nose pliers; 4. small screwdriver; 5. ink rubber.

If you have any keyswitches that are hard to use on your MicroBee and you can't afford the down-time to send it in for repairs, then you or someone you know who can use a soldering iron should be able to fix that troublesome keyswitch.

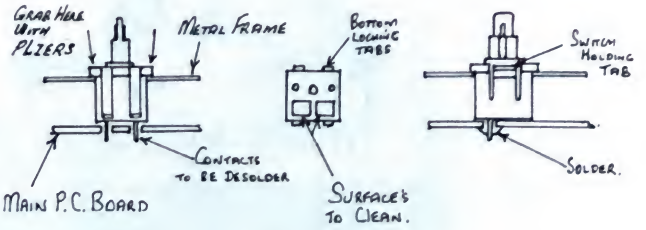
The keyswitch has to be removed and the contacts inside cleaned with an ink rubber or very fine emery paper.

To remove a keyswitch the MicroBee has to be completely removed from its covers. Remove the keytop of the faulty key – just pull it straight off. Then, with your soldering iron and solder sucker, remove the solder from the holes holding the keyswitch in the main board. **Caution:** do not use too much heat on the switch contacts or you may melt the plastic switch case.

With the solder removed, from the keyboard side grab the keyswitch body with a pair of long-nose pliers near the frame on the right and left sides. Then pull up; if all the solder has been removed it should pull out easily.

Then with a very small screwdriver remove the bottom of the switch (**watch out for the spring**). With the switch apart clean the two gold contacts. Re-assemble the switch and install back into the Bee.

The whole job will take around 15 minutes. **Note:** only attempt this on machines out of warranty. Keyswitches are completely replaced on warranty machines or any machine serviced by Applied Technology.



CASSETTE MOTOR CONTROL FOR LOADING AND SAVING ON MICROBEES WITH COLOUR 5.22e BASIC

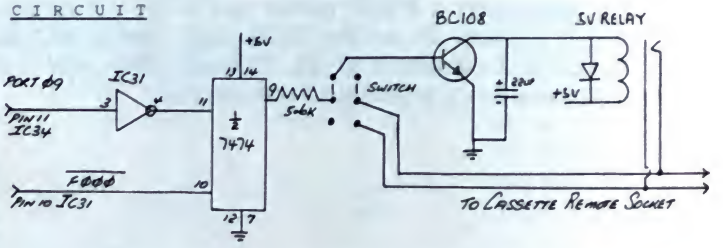
S O F T W A R E

```
Color Basic LOADING      > LOAD (cr)
   "      "  SAVING      > OUT 9,0 : SAVE "test" (cr)
```

H A R D W A R E

1. 7474 or 74LS74
1. 5.6K Resistor
1. 1N914 Diode
1. D.P.D.T. Toggle Switch
1. Minature 5V Relay
1. BC108 Transistor
1. 22uf Tag Capacitor

C I R C U I T



On the IC MicroBee the 7474 can be placed in IC32 position (cut tracks first), connect pin 1 IC30 to 0V. The relay can be mounted in the spare socket.

Operation

Out 9,0 set the 7474; this latches the relay when the contacts close, starting the motor. When anything is written to the screen F000, the 7474 is reset, stopping the motor. During loads and saves nothing is written to the screen except the "*". The capacitor stops the relay from changing during this write to the screen. Load and Save out 9,0, then write and set the 7474 again. PLAY, LLIST, LPRINT also out 9,0, so the switch has been added to stop the relay clicking during play. This allows you to rewind your tapes without disconnecting any plugs. Test control "G" then space.

CASSETTE MOTOR CONTROL FOR LOADING AND SAVING ON MICROBEES WITH STANDARD 5.1 BASIC

Note: With 5.1 BASIC you can use the same circuit as the Colour BASIC but you cannot use the $\overline{F000}$ signal and you have to enter a bit more software.

Software

5.1 BASIC LOADING > OUT 9,0 : LOAD (cr)

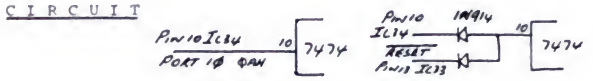
When the tape is loaded use one of the two suggestions in the operation.

5.1 BASIC SAVING> OUT 9,0 : SAVE "test" : OUT 10,0 (cr)

Or> OUT 9,0 : FOR X=1 to 3: SAVE "test" :next x : OUT 10,0 (cr) saves "test" three times then stops motor.

Hardware

Same as used with Colour BASIC version.



The 7474 can be mounted in the spare socket but the relay will have to be mounted where it fits best. Stick it on the back or side near the spare socket.

Operation

To have the motor stop automatically after loading, save your programs with Auto start; the first line of the program could be 100 OUT 10,0 REM stop Motor on Cassette.

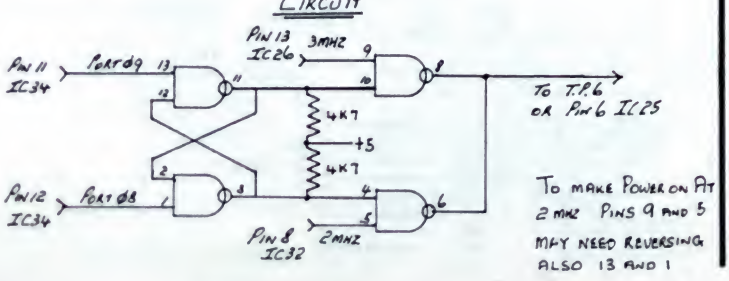
Use port 10 not port 8; if your programs are run on a Colour MicroBee they will not upset the colour. Port 8 is used for colour.

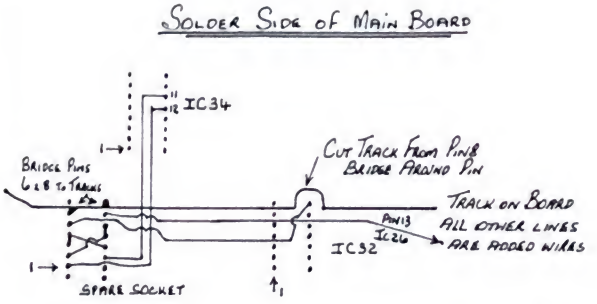
The other way to stop the tape is to manually press Reset or type in ›OUT10,0.

DUAL-SPEED MICROBEE WITH 5.1 BASIC

Change Clock Speeds
From 2 MHZ to 3 MHZ
using Software (no switches)

Hardware Required 1 x 7403 2 x 4.7K resistors
Z80-A Z80 PIO-A





SOFTWARE CHANGING SPEED IN PROGRAMS

```
10  FOR A = 1 TO 24 : PLAY A : NEXT A
20  OUT 9,0 : REM SWITCH TO 3 MEG
30  FOR B = 1 TO 24 : PLAY B : NEXT B
40  OUT 8,0 : REM SWITCH BACK TO 2 MEG
50  GOTO 10 : REM LOOP
```

DECODING EXTRA INPUT/OUTPUT PORTS

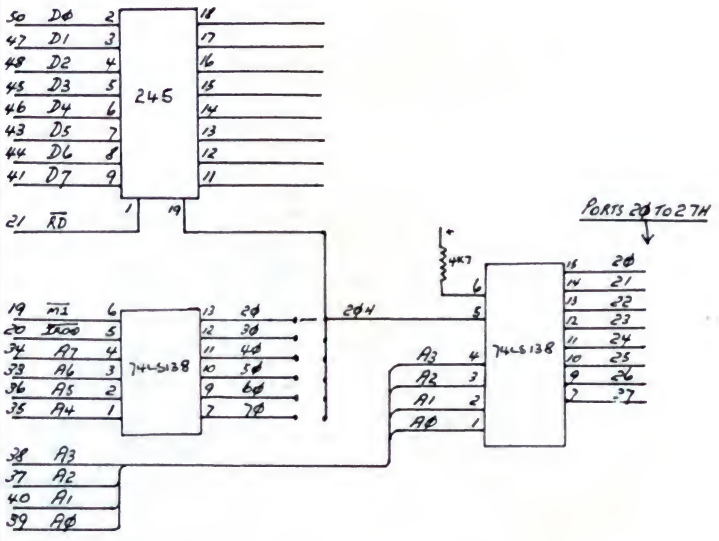
Ports 20 Hex to 27 Hex or 2F hex
Selectable from 20 Hex to [illegible] Hex in 10H steps

Note:

Ports 00 to 20 hex are decoded in the MicroBee
Port 00 is the same as 10H
Port 08 is the same as 18H
Port 0F is the same as 1FH

Address line A4 is not decoded, causing the double up. Not to worry, 223 ports left.

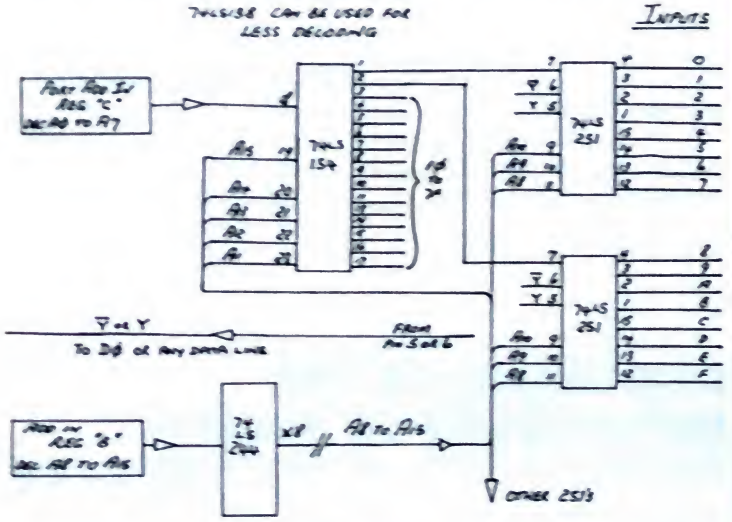
Address lines A0 to A7 and control signals $\overline{IORQ}$, $\overline{M1}$ and $\overline{RD}$ need not be buffered if they are only used to drive one gate each. Data lines must be buffered.



Note: By replacing the 74LS138 with 74LS154 you can decode twice as many ports. See data books for pin connections. ▶

SCANNING UP TO 255 SEPARATE INPUTS ON ONE PORT

Using Machine code to check for on/off, zero or one.




```
ADDR   CODE     LINE   LABEL   MNEM   OPERAND

3000            00100          ORG    3000H
3000 2A0040     00110          LD     HL,(4000H);Address to store results
3003 01200F     00120          LD     BC,0F20H  ;Port20 C  start B at 0F
3006 EDA2       00130          INI              ;input with increment
3008 C9         00140          RET
0000            00150          END
00000 Total errors
```

For output use the same decoding circuits, but the output chips will be different. 74LS154 or 74LS138 if no latching is required. If latching is required use 9334, 74LS259 or NE5090.

There are plenty of output instructions to use using Reg 'B' as a counter of A8 to A15.

64K ROM PACK

This pack will work on any 16K or 32K MicroBee with 5.1 BASIC or 5.22e Colour BASIC. All that needs fitting to the MicroBee is a 50-way expansion plug to the core board; no modifications are needed.

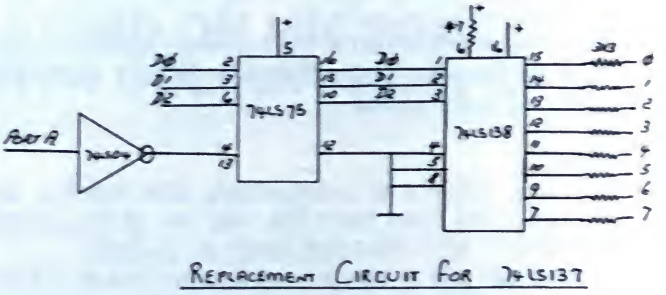
ROMs or the ROM at COOO and DOOO in the MicroBee must be removed and fitted in the pack. These ROMs cannot be disabled under software without modifications.

To use the pack with 5.1 BASIC, enter under BASIC 10, (n), where n is any number from 0 to 7. Then enter >EDASM, and ROM at location (n) will be executed.

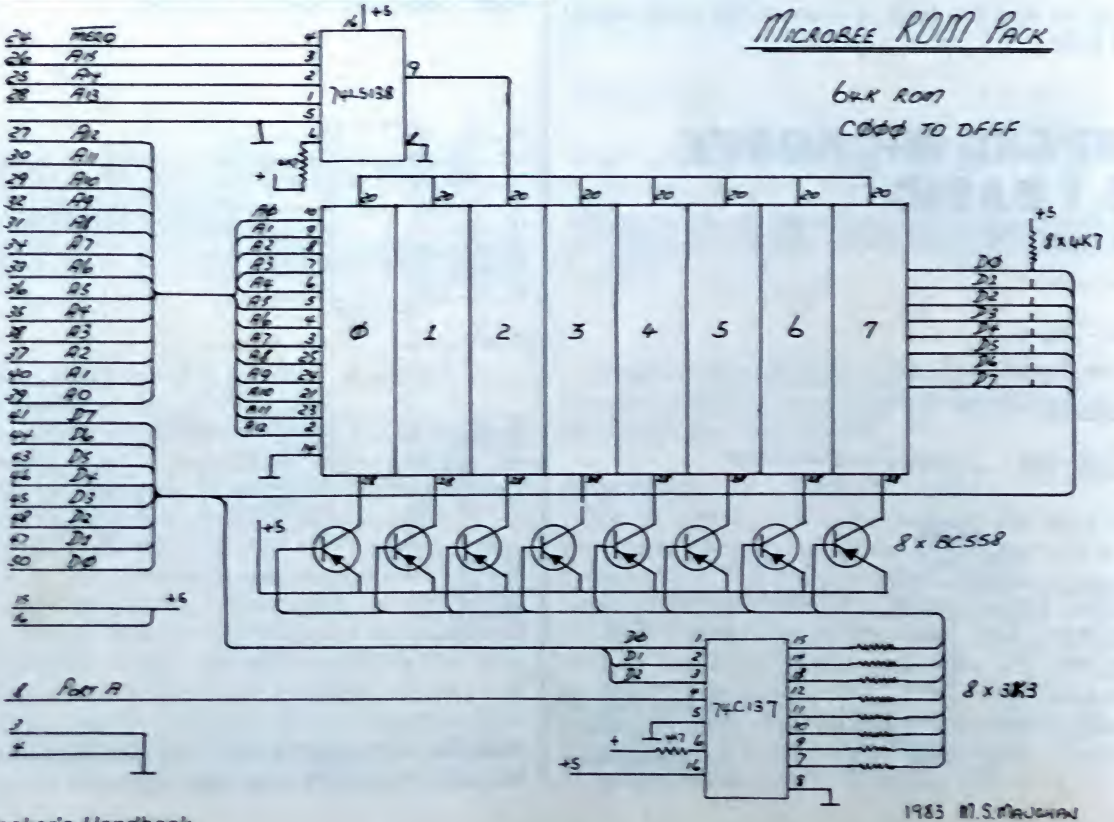
With the IC MicroBee enter under BASIC >PAK (n) where n is any number from 1 to 7. (ROM 0 is in the Bee and has to be removed or the core board modified.) The ROM at location (n) will auto execute. To use ROM 0 use ›OUT 10,0 then PAK.

The eighth set of 8K ROMs are all wired in parallel except for the power supply pin to +5 volts. The five volts is only applied to the selected ROM and is latched there until a new ROM is selected. You can only have one ROM powered up at any one time. There is no need for extra power supply to run this board; it uses the +5 V from the core board.

If you cannot get a 74C137 IC then the following circuits will do the same job.



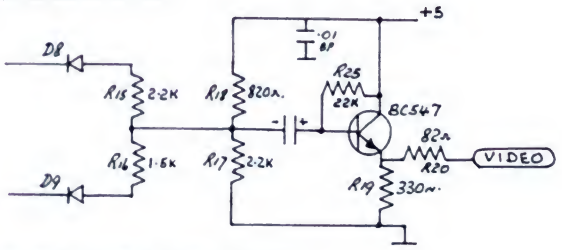
REPLACEMENT CIRCUIT FOR 74LS137



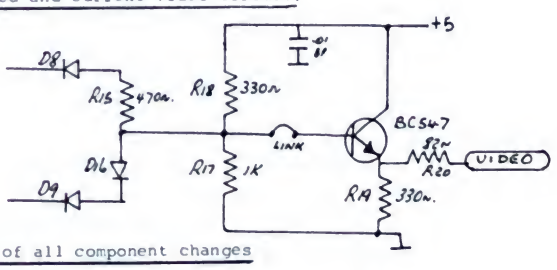
1983 M.S.MAUGHAN

CIRCUIT CHANGES (ONLY FOR EARLY MICROBEES)

Orginal Video Circuit.



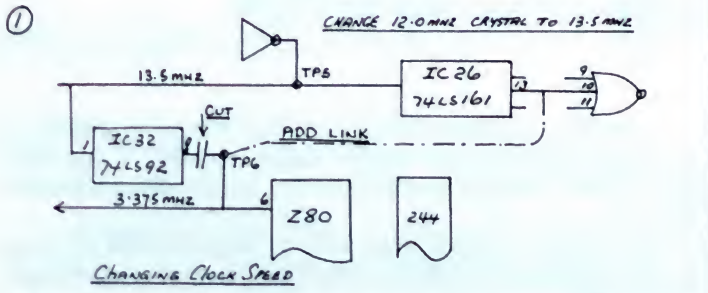
Updated and current Video Circuit.

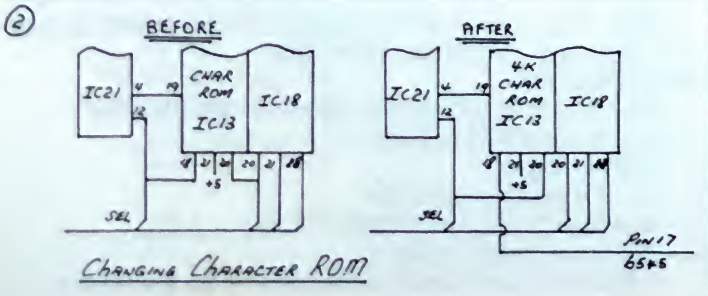


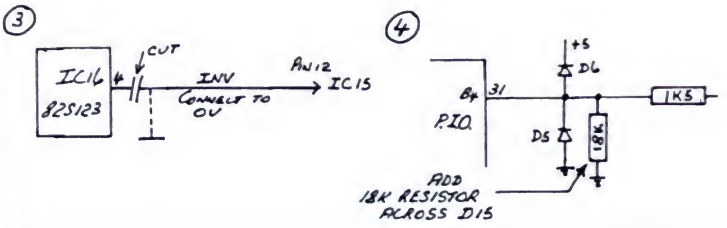
List of all component changes

	WAS	NOW
R1	3.3K	10K (CTS)
R15	2.2K	270 ohms
R16	1.5K	1N914 now D16
R17	2.2K	1K
R18	820 ohms	330 ohms
R19	330 ohms	330 ohms no change
R20	82 ohms	82 ohms no change
R21	4.7K	82 ohms (tape circuit)
R25	22K	NOT USED
C20	22 uf	NOT USED
D5	1N914	18K(s red in parallel with DIODE)

UPGRADING THE BASE BOARD







Note: Base boards from REVISION H and later have modifications (2) and (3) already done. The artwork has been changed. Check board before doing mods 2 and 3.

OFFICIAL STANDARD FOR JOYSTICKS ON THE MICROBEE

DB15 port and joystick connections.

Connections for the Spectravideo 'Quick Shot' joystick. This unit normally has a DB9 plug fitted and for the MicroBee you need a DB15.

Connections for the DB9:

1. UP
2. DOWN
3. RIGHT
4. LEFT
5. Not Used
6. FIRE
7. Not Used
8. Switch common
9. Not Used

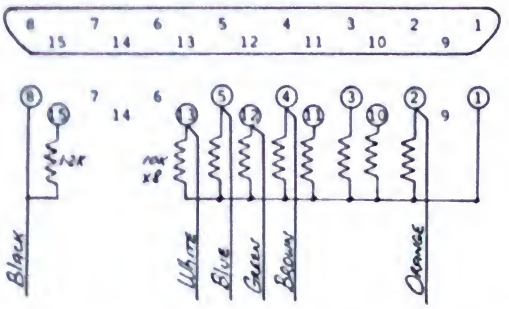
Colour code of wiring:

White = UP, Blue = DOWN, Brown = LEFT, Green = RIGHT, Orange = FIRE, Black common return for all switches.

Connections For MicroBee Via The DB15

Pin	Status	Usage	DB15 connections All 10k Resistors	
1	Supply +5	Common for Resistors	All 10k Resistors	
2	Data 7	FIRE button	Orange	+ 10k
3	Data 5	Select player 2		+ 10k *
4	Data 3	LEFT	Brown	+ 10k
5	Data 1	DOWN	Blue	+ 10k
6	Not used		N.C.	
7	A ready	Do not connect	N.C.	
8	Earth 0v	Return for Switches	Black	
9	Not used		N.C.	
10	Data 6	Not allocated		+ 10k *
11	Data 4	Select player 1		+ 10k *
12	Data 2	RIGHT	Green	+ 10k
13	Data 0	UP	White	+ 10k
14	Not used		N.C.	
15	A strobe	Pull low (to pin 8)	1k2 resistor to pin8	

The three pull-up resistors marked with an asterisk are not essential, but help to give reliable input from the port. N.C. means Not Connected.



Pinout of the DB15 connector viewed from the solder side.

TESTING YOUR JOYSTICK

When you have modified or constructed your joystick, you will need to test it and check it is functioning correctly.

```
00100 REM joystick test program for Microbee
00110 OUT 1,255              :REM initialise the port
00120 A = IN(0)              :REM read joystick on port zero
00130 A = 143 - (A AND 143)  :REM converts to positive logic
00140 B = -(A AND 1)    :IF B THEN PRINT "up",
00150 B = -(A AND 2)    :IF B THEN PRINT "down",
00160 B = -(A AND 4)    :IF B THEN PRINT "left",
00170 B = -(A AND 8)    :IF B THEN PRINT "right",
00180 B = -(A AND 128) :IF B THEN PRINT "fire",
00190 IF A=0 THEN PRINT "nothing",A ELSE PRINT A
00200 FOR T = 1 TO 100 : NEXT T : REM short delay
00210 GOTO 120
```

If the program is RUN without a joystick connected, the display:

up down left right fire 143

should print continuously up the screen (provided all inputs of your PIO float LOW with no input connected). When the joystick is plugged in, the display should change to:

nothing 0

repeating continuously. If you now push the joystick forward the display should change to:

up 1

Now press one of the fire buttons while still holding the stick forward; this should show:

up fire 129

Move it slightly left and you should see:

up left fire 133

Continue until all combinations have functioned correctly. Once you are satisfied that all is functioning correctly you can start writing your own joystick software, or modifying your existing programs to make use of the joysticks.

Don't forget if you develop any good programs please take them to your local MicroBee distributor for forwarding to 'Honeysoft', or send them directly to:

Honeysoft Pty Ltd
82a Jubilee Tce
Bardon 4065 Qld

We may be able to sell them on your behalf, or at least include them in users' group material for distribution to other MicroBee owners.

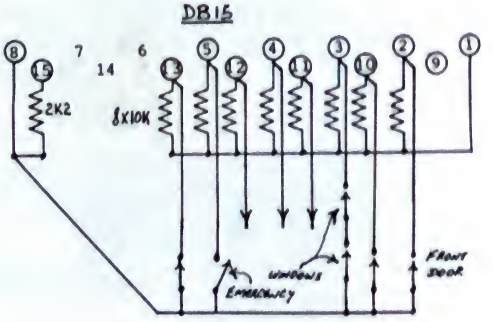
SIMPLE INTRUDER ALARM

One way to use your MicroBee for a simple alarm system is to use the parallel port. Wire it up in a similar way to the joysticks, only instead of wiring up to the joystick switches wire up the reed switches and contacts on your windows and doors.

The program will have to look for open circuit and not closed circuits, but this shouldn't cause any problems. Exit and entry delays can also be programmed. The alarm can be the internal speaker or use a spare port – 9 or 10. If you have the cassette load control relay fitted, use this to drive an external alarm (with its own circuit and power supply). The length of time the alarm is to ring can also be programmed, as can self-testing.

This is one way of getting your MicroBee to work for you when you're not using it. Switch your screen off; it's not needed when you're not looking at it. Just switch it on to check the status of the doors and windows when the program is run.

Leaving your MicroBee on 24 hours a day won't hurt it; there is very little to wear out.



With eight inputs you can have a good selection; window switches can be wired in series on one bit. You can also program to look for closed circuits such as emergency switches.

MODIFYING A TV SET FOR USE AS A MONITOR

To achieve the best results from your MicroBee with its 64 characters per line you need a special monitor. If you can't afford one of these the next best thing is a modified television set.

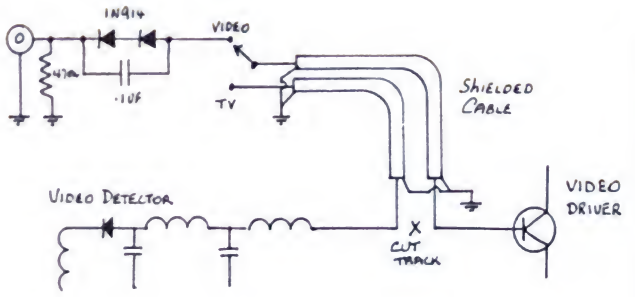
If you have a small B/W portable or large old B/W set, then in most cases it is possible to use them as a switch-selectable TV/monitor.

Note: There are some TV sets you cannot convert, so a circuit diagram of your set is needed first.

Parts Required

1 x RCA socket to connect the video from the MicroBee
1 x 470 ohm 1/4 watt resistor
2 x 1N914 diodes
1 x 0.1 uf monolithic capacitor
1 x spdt switch (if you still need a TV set)
2 x 600 mm shielded cable

All these can be mounted in the set near the aerial input.



Note: You cannot expect to get the same quality picture from a converted TV as you get with a professional monitor. If you want to do a lot of computing a good monitor is the best investment.

INTERFACING THE HALF-INTENSITY COLOUR SIGNALS

With three colour signals, R G B, it is only possible to get eight foreground and eight background colours. Black is counted as a colour.

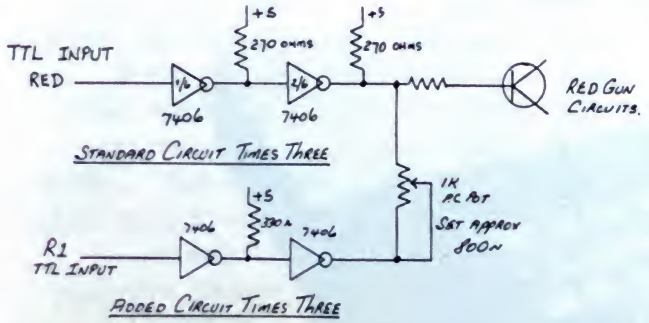
To get a greater range of colours the MicroBee has three extra signals, R1, G1, B1; these are the half-intensity signals. All six signals are TTL level signals (you can't have half-level TTL). Mixing R with R1 has to be done in analogue and is best done in your monitor.

Most colour monitors accept TTL level R G B signals, then convert them to analogue to drive the colour guns. The half-intensity signals have to be interfaced between the TTL output (in the monitor) and the analogue input.

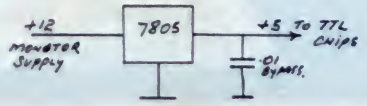
The following circuits show one possible way to do this mixing; note: it may not be possible with all monitors.

MICROBEE DB15 PIN CONNECTIONS

1. C.SYNC +	2. C.SYNC –	3. B2	4. G2	5. R2	6. Blue
7. Green	8. Red	9. 0 volts	10 to 15	0volt GND	



5VOLT POWER CIRCUIT MAY NEED CHANGING TO POWER EXTRA IC



WORDBEE TAB SCALE

```
00100 CLS REM  M.S.Maughan 7th July 1983
00110 IF PEEK (2225)= 1 THEN 190
00120 UNDERLINE
00130 CURS207 :PRINT"W o r d b e e   T a b   S c a l e"
00140 NORMAL
00150 POKE220,20 :POKE140,0 :POKE216,17
00160 POKE162,30 :POKE163,128 :POKE2225,1
00170 CURS340 :PRINT "Press Reset to Continue"
00180 GOTO180
00190 FOR X=62464 TO 62527 :POKE X,46 :NEXT X
00200 Y=49
00210 FOR X=62464 TO 62527 STEP 8
00220 POKE X,Y : Y=Y+1 :NEXT X
00240 EDASM
00250 END
```

HEX-DECIMAL CONVERSION TABLE

COL. 3		COL. 2		COL. 1		COL. 0	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

```
TO  convert  8517 Hex to Decimal
          col. 3     8 =   32,768
          col. 2     5 =    1,280
          col. 1     1 =       16
          col. 0     7 =        7
                  ----------------
                      34,071  or 34071 decimal
```

PROGRAM AID VARIABLES USED

VARIABLES USED IN PROGRAMMING.....PROGRAM

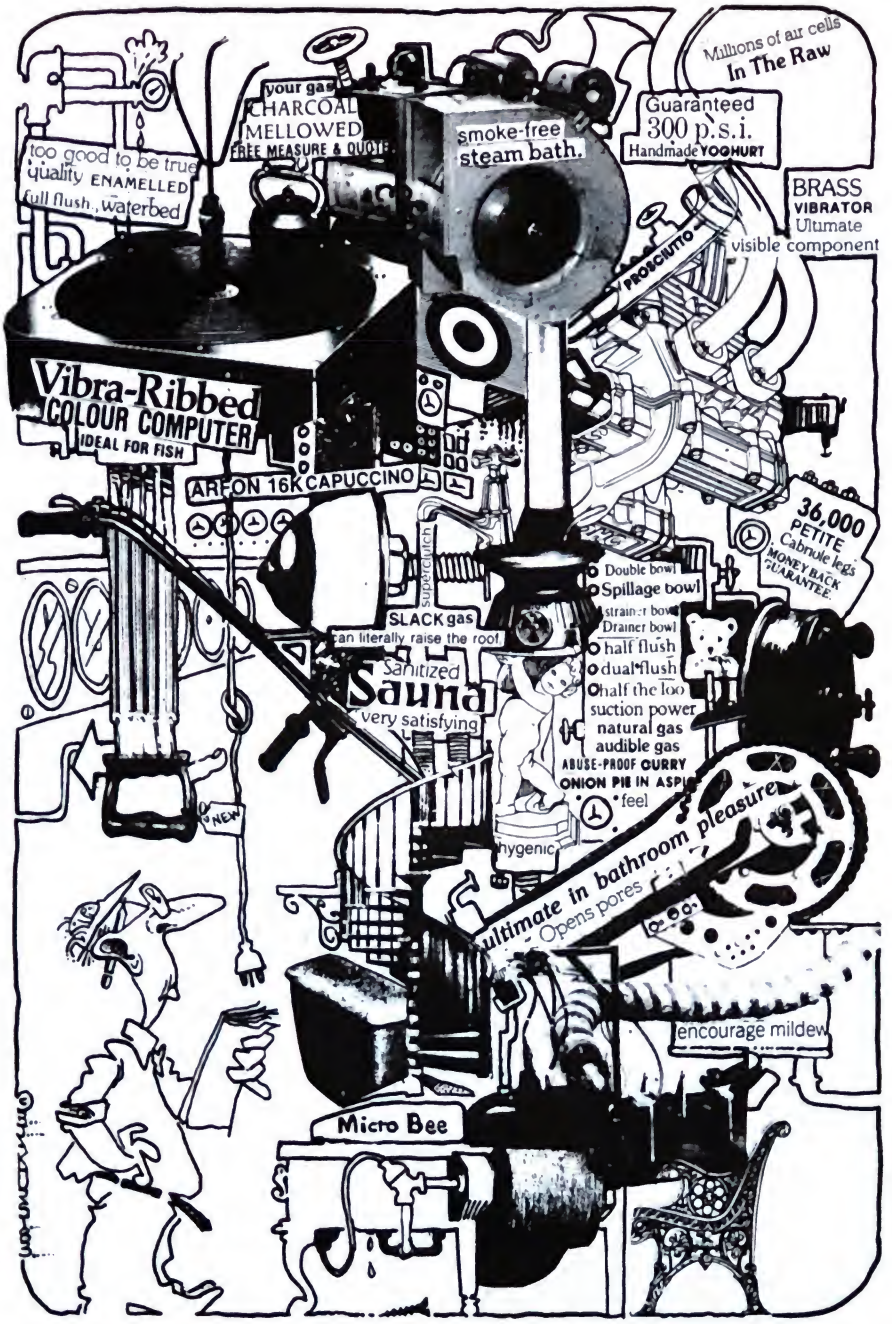
A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	
A0	A1	A2	A3	A4	A5	A6	A7	
B0	B1	B2	B3	B4	B5	B6	B7	
C0	C1	C2	C3	C4	C5	C6	C7	
D0	D1	D2	D3	D4	D5	D6	D7	
E0	E1	E2	E3	E4	E5	E6	E7	
F0	F1	F2	F3	F4	F5	F6	F7	
G0	G1	G2	G3	G4	G5	G6	G7	
H0	H1	H2	H3	H4	H5	H6	H7	
I0	I1	I2	I3	I4	I5	I6	I7	
J0	J1	J2	J3	J4	J5	J6	J7	
K0	K1	K2	K3	K4	K5	K6	K7	
L0	L1	L2	L3	L4	L5	L6	L7	
M0	M1	M2	M3	M4	M5	M6	M7	
N0	N1	N2	N3	N4	N5	N6	N7	
O0	O1	O2	O3	O4	O5	O6	O7	
P0	P1	P2	P3	P4	P5	P6	P7	
Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	
R0	R1	R2	R3	R4	R5	R6	R7	
S0	S1	S2	S3	S4	S5	S6	S7	
T0	T1	T2	T3	T4	T5	T6	T7	
U0	U1	U2	U3	U4	U5	U6	U7	
V0	V1	V2	V3	V4	V5	V6	V7	
W0	W1	W2	W3	W4	W5	W6	W7	
X0	X1	X2	X3	X4	X5	X6	X7	
Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	
Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7	

50-WAY EXPANSION PORT CONNECTIONS

PIN	Z80 NAME	DESCRIPTION
1,2	+12 v Input/output	Power connection for Microbee
3,4	0v	system ground
5	Port-09 * (O)	goes low when port is accessed
6	Port-08 * (O)	goes low when port is accessed
7	WAIT * (I)	used to slow down the CPU
8	Port-0A * (O)	goes low when port is accessed
9	INT * (I)	interrupt input
10	HALT * (O)	indicates CPU is halted
11	NMI * (I)	generate non maskable interrupt
12	CPU clock (O)	3.375 mhz or 2 mhz (N1)
13	BUSAK * (O)	Acknowledge bus request
14	BUSRQ * (I)	request use of bus
15,16	+5 regulated from CORE BOARD	
17	RFSH * (O)	for refresh on dynamics RAM
18	RESET * (I/O)	O/C input/output
19	M1 * (O)	indicates instruction fetch
20	IORQ * (O)	indicates port access
21	RD * (O)	read strobe
22	XWR * (O)	deglitched write strobe
23	SEE NOTE N2	see note (N2)
24	MREQ * (O)	indicates memory access
25	A14	
26	A15	Most significant address BIT
27	A12	
28	A13	
29	A10	
30	A11	
31	A8	
32	A9	
33	A6	
34	A7	
35	A4	
36	A5	
37	A2	
38	A3	
39	A0	
40	A1	
41	D7	128 data line
42	0V	system ground
43	D5	32 data line
44	D6	64 data line
45	D3	8 data line
46	D4	16 data line
47	D1	2 data line
48	D2	4 data line
49	0V	system ground
50	D0	1 data line

Notes:

* means active low signal
(O) means output from MicroBee
(I) means input to MicroBee
(N1) Standard MicroBees = 2 MHz. Early 64K = 2.25 MHz. IC models = 3.375 MHz
(N2) Core boards, old style 8000 series – this is (O) ROM SELECT * ; new style core boards and 56K board – this is PHANTOM (I) *



ADDING THINGS ON

Haven't you always thought that your Microbee could be the very best machine ever ... if only it had a proportional analogue joystick? ... (hammer, hammer, hammer) ... And if it had a parallel printer interface ... well, it follows that you could hang a parallel printer off the side, doesn't it? ... (bash, bash) ... And that printer would really be earning its keep if you could somehow wire up a phase-locked loop decoder, a pitchpipe tuning aid and a shortwave receiver, to the Bee, so you could receive the signals that would let you print out weather maps ... a bit of Clag should hold it ... hmm ... There must be room for a ... um ... ROM reader to plug in the back there somewhere ... Oh, but there's an 8-bit port free, too ... a super special 'screen spotter' light pen would be just the thing ... dum-de-dum ... Need a bit more ROM room – just daisy-chaining a few slave boards will give you up to 308K total ... pass the, er, daisies would ya? ... whistle while you work ... Couldn't possibly fit a modem on there – too expensive to run anyway – so ... ah ... why not ... rrrip out that Telcom EPROM ... and bung in a new one that'll make the Bee display in a different typeface? ... piecacake ... time for a quick game before lunch ... hmm ... might just pop in an Extended CHIP-8 V2.0 interpreter/compiler from Dreamcards ... makes things so much more spectacular, don't you think? ... Zoink, Zap, Neeeooow, Vroom, Pop, Frizzle ... Wow, got the munchies something fierce ... Can't ... seem ... to find ... the door ...

Proportional analogue joystick for the Microbee

The joy of a joystick! A twiddle of the fingers as your spaceship races through the cosmos, zapping space invaders and other various baddies with searing photon torpedoes. Or a tank on a battlefield, dodging the mines. Or your Tiger Moth bucks through the turbulence as your skilful fingers guide it gently towards a landing. Fasten your seatbelts, folks, because with this ETI project, you'll be joining the fun.

Geoff Nicholls

Tom Moffat

MOST COMPUTERS offer joysticks as options. And most of them aren't as snazzy as this one. Your average home computer joystick, or games controller', is nothing more than four switches, actuated by a central handle. Leaning the handle one way or another actuates the appropriate switch, which tells the computer to drive a dot or other shape across the screen in the direction given. Some computers allow two switches to operate at once; that is, if you lean the handle 45 degrees from North, the North and East switches will operate together, moving the dot from lower left to upper right.

This arrangement is quite OK for many applications, but you can only move at one speed, in only eight different directions. Now, consider the true analogue joystick. On the outside it looks the same . . . a vertical handle that can be moved around. BUT . . . when you move the stick say 24 degrees from North, the dot will follow exactly. If you move quickly the dot moves quickly. If you move the stick slowly the dot just creeps along, and it stops or changes direction when you do. Here's where the really subtle control becomes available for such tasks as a precision landing of a lunar module. You can even write your name with it.

All analogue joysticks must use an analogue-to-digital converter. Many computers have them built in but the Microbee doesn't, so we have to supply one. Two, actually, one for vertical values and one for horizontal. Six-bit conversion is used, giving 64 different values each way. The seventh bit tells the joystick whether an X or a Y value is required, and the eighth bit feeds results back into the computer. So, the whole business works through one eight-bit port.

A bit approximate

Before the joystick can be used, a short driver routine must be loaded into the computer, usually as part of the main program requiring the joystick. The program, and the joystick A/D converter, form a device called a "successive approximation converter". It is a very fast scheme that works in the following way:

Imagine you have a piece of coaxial cable, and some ratbag has stuck a pin in it, and broken the pin off. You know the cable is shorted, but you don't know where. There are two ways to find out . . . first you can cut off a metre at a time until it comes good. If luck isn't with you you will end up with a nice collection of one metre lengths of coax, with the short in the very last one. (Shades of Monty Python's "stringettes" — 3" lengths of string, good for tying small parcels, ideal present — Ed.)

In the "sensible" method you cut your losses, so to speak. You first cut your coax in half, knowing that will leave one good bit and one bad bit. You then cut the bad bit in half, leaving half of that good and half bad. You keep on cutting the bad bits in half until you have localised the short. The beauty of this system is that no matter where the short is, it takes exactly the same number of cuts to find it. And it's the fastest way to find the short.

In the joystick we cut its value in half, and then look at a comparator. If the value is above half the maximum it tells the computer to store a bit, and then cuts the upper half in half, looking for a "too high" or "not high enough" indication. And so it goes for all six bits. The result is a binary representation of the joystick value, stored in the computer.

The "halves" are generated by that

HOW IT WORKS — ETI-674

The circuit includes most of a successive approximation analogue-to-digital converter. The rest of the converter is actually the Microbee!

Resistors R4 to R22 form a binary ladder network, also called an 'R-2R' network. The actual resistance of the resistors in the network is not critical, provided they are closely matched. IC4 is used to ensure that the digital signal levels from the computer's PIO will swing to within millivolts of the supply rails, i.e: 0 or 5 volts. If the six-bit number applied to IC4 is 'n', then the voltage at the junction of R4-R6 will be Vcc x n/96. Thus, if n=0 the voltage will be 0 and if n=63 the voltage will be 3.28 volts.

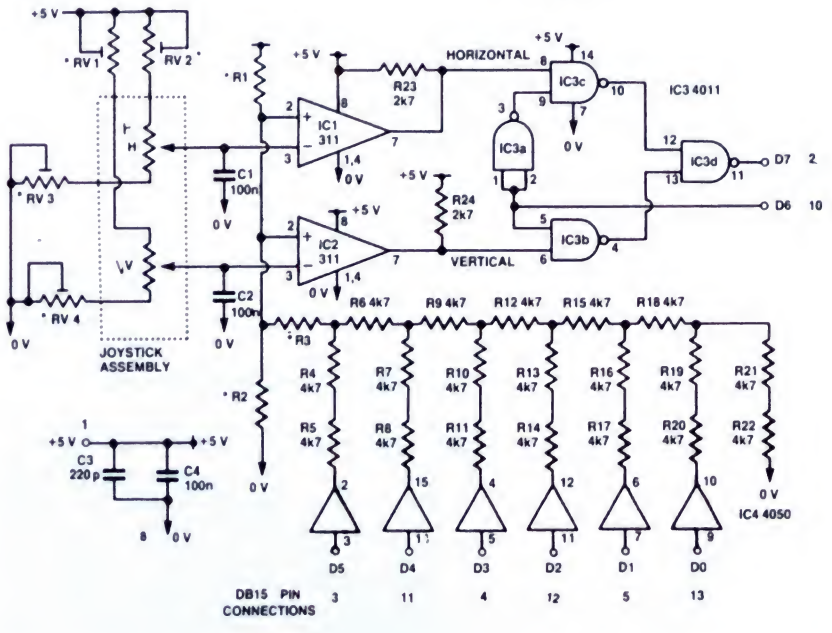
Resistors R1, R2 and R3 are chosen to shift the voltage range to match the output voltage range from the joystick. Trimpots RV1-RV4 are used to fine-tune the joystick range.

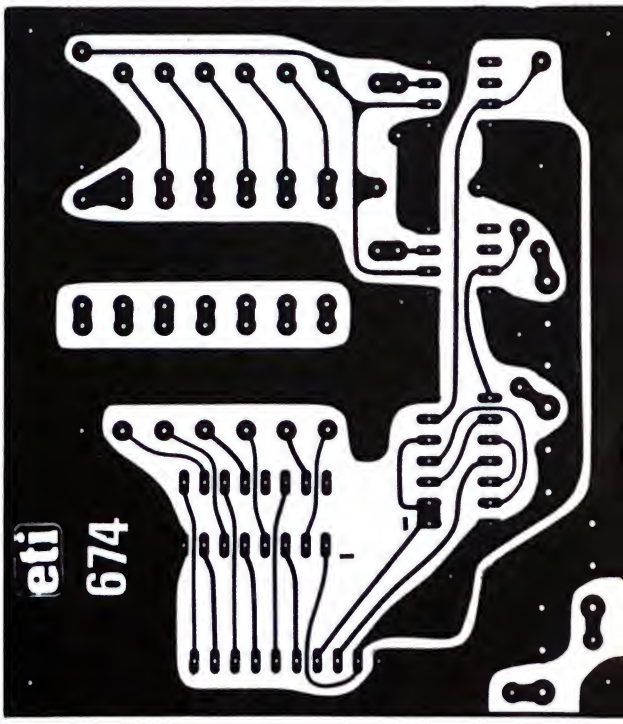
Comparators IC1 and IC2 compare the voltage from the joystick ports to the voltage from the ladder network; if the latter exceeds the former, then the comparator output is high, and vice versa. IC3 selects either the horizontal or vertical comparator output to feed to the computer, according to the state of bit 6, which is set by the software as required.

To read the joystick, the computer selects the pot (horizontal or vertical), then starts outputting six-bit data via D0-D5, while reading the comparator output. As outlined in the text, this process is done one bit at a time, so only six operations are required, independent of the actual value. This technique is called successive approximation.

```
ADDR    CODE LINE    LABEL   MNEM     OPERAND

                00100 ;JOYSTICK DRIVER ROUTINES, Tom Moffat, 1/12/82
                00110
0400            00120          DEFR     16        ;ASSUME HEX VALUES
0800            00130          ORG      0800
                00140
                00150 ;Initialize the PIO:
                00160
0800 3ECF       00170          LD       A,0CFH    ;SET FOR CONTROL
0802 D301       00180          OUT      (1),A
0804 3E80       00190          LD       A,80      ;DDR 10000000
0806 D301       00200          OUT      (1),A
0808 C9         00210          RET
                00220
                00230 ;Joystick routine, get X if C=00, get Y if C=40.
                00240
0809 79         00250          LD       A,C       ;SELECT X OR Y (BIT 6)
080A 1620       00260          LD       D,20      ;SET "TRY" BIT (BIT 5)
080C B2         00270 LOOP     OR       D         ;COMBINE THE TWO AND...
080D D300       00280          OUT      (0),A     ;SEND TO A/D CONVERTER.
080F DB00       00290          IN       A,(0)     ;GET THE RESULT.
0811 CB7F       00300          BIT      7,A       ;TEST COMPARATOR BIT.
0813 2801       00310          JR       Z,$+3     ;IF LOW SKIP NEXT INSTR.
0815 AA         00320          XOR      D         ;KILL "TRY" BIT IN A REG.
0816 CB3A       00330          SRL      D         ;SHIFT TO NEXT POSITION.
0818 30F2       00340          JR       NC,LOOP   ;DO AGAIN UNTIL FINISHED.
081A E63F       00350          AND      3FH       ;CHOP GARBAGE OFF FRONT.
081C 4F         00360          LD       C,A       ;LOAD WHAT'S LEFT INTO C.
081D C9         00370          RET                ;AND RETURN TO BASIC.
0000            00380          END
00000 Total errors

LOOP    080C
```

LISTING 1





great network of 4k7 resistors, with the output feeding two comparators at once. Each joystick pot, X and Y, goes to one comparator. A series of NAND gates selects which comparator output goes back to the computer. The NAND gate X/Y selector is driven by bit 6 from the computer. An assembly language listing of the A/D converter driver is provided for those interested.

There are actually two subroutines necessary for the joystick to be used in a BASIC program. First the routine at 2048 must be called to set up the parallel port for the joystick. This need be done only once at the start of the program. Then the routine at 2057 is called to get a joystick value. The "USR" function in Micro-World BASIC makes this easy, since you can pass information both to and from a machine code routine. You send in a "0" to get a horizontal value, or a "64" to get a vertical value. To specify a point on the screen you must do each one.

The BASIC demonstration program given in Listing 2 shows how this works. The program only writes dots to the screen, leaving a trail of dots behind. Anything more elegant must come from your own imagination. ▶

Joystick types

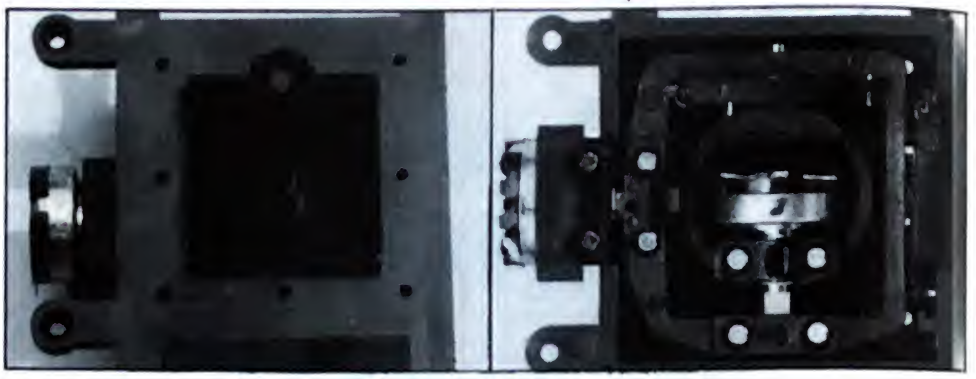
There are a number of different joystick assemblies around, and the type you use affects some components in the circuit. The important parameters are;

(1) Ratio of minimum resistance to total resistance.

(2) Ratio of maximum resistance to total resistance. The project has been built up twice, once by Tom Moffat and once by ETI. Tom used a joystick with pots of about 100k total resistance, with minimum and maximum resistances of 0 and about 90k.

The ETI version used pots with minimum resistances of around 8k and maximums of around 12k out of a total resistance of 20k. We understand that the most common type supplied will be mechanically identical to ours but will have a total resistance of around 5k. This means that the ratios mentioned earlier will be the same as ours, but the trimpots would be reduced from 5k to 2k to give a useful range adjustment.

The table here gives the component values to suit the three types of joystick discussed.



Close up. Front and rear views of the joystick used.

Construction

First work out where the joystick will mount in the box, remember that the pc board has to fit as well! Put the joystick right up one end so that a large area is left to rest the wrist on — you don't want to develop an ache after a few minutes use. A square cut-out was required for our stick (obtained from Benelec). Drill lots of holes and file out to exactly fit the flange on the stick. The unit is secured with four self-tappers through the box to bite into the plastic mounts on the stick base. Solder wires to each lug on the pots, around 200 mm allows slack when the pots are moved.

File a bevel to allow the ribbon cable to pass between the box and its lid, and the mechanical stuff is done.

The pc board can be made up now, start with the 4k7 resistors, and be careful not to put them where R2 and R3 have to go. Solder in the remaining resistors and the four capacitors and trimpots. The ribbon cable comes next, if you are using a solder-type DB15 plug then simply follow the overlay diagram. Some suppliers may provide insulation-displacement (ID) plugs, you'll have to carefully work out the wires if you get one — watch the 0 V wire, it is out of sequence.

The wires from the joystick pots can be soldered in now. Next, solder the ICs in place. Take particular care to orient them correctly and avoid touching the pins of IC3 and IC4 — they are static sensitive. Solder their Vcc and 0 V pins first.

The pc board was held in place with double-sided sticky pads, one also being used to secure the ribbon cable to the inside end face of the box.

JOYSTICK	R1	R2	R3	RV1,2	RV3,4
5K	100K	82K	180K	2K	2K
20K	100K	82K	180K	5K	5K
100K	none	91K	91K	20K	none

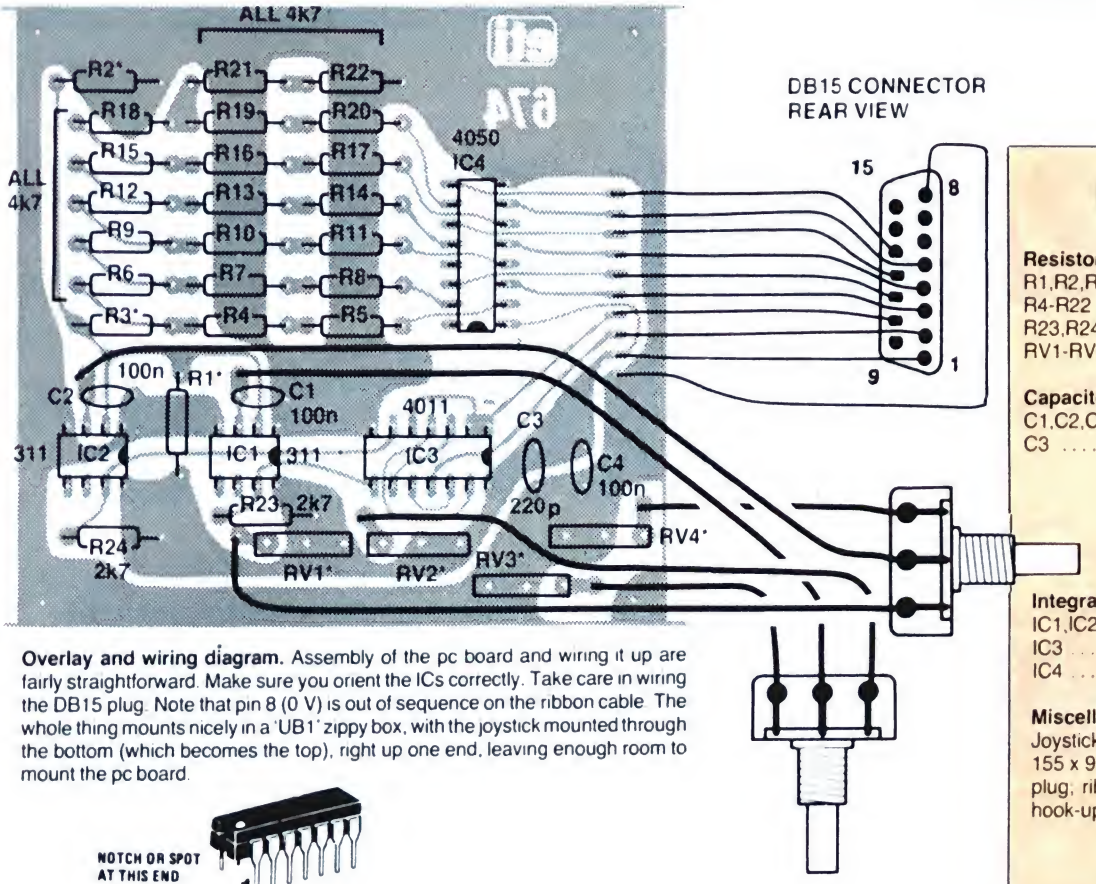
Note that, when using the 100k version the 0 V wires from the pots have to return to 0 V on the PC board.

LISTING 2

```
00100 REM Test programs for ETI 674 joystick project.
00110 REM
00120 REM Initialize
00130 GOSUB 450
00140 REM
00150 PRINT"1 : A-D test"
00160 PRINT"2 : Range test"
00170 PRINT"3 : Drawing"
00180 INPUT"Program no.?";P
00190 IF P<1 ORP>3 THEN 180
00200 CLS:ON P GOTO 220,300,380
00210 REM
00220 REM #1 : A to D converter test.
00230 PRINT "Measure voltage at pin 2 of IC1"
00240 PRINT "Type test no.s from 0 to 63."
00250 PRINT "To exit type 64"
00260 INPUT A: IF A>63 THEN STOP
00270 OUT 0,A
00280 GOTO 260
00290 REM
00300 REM #2 : Range adjustment program.
00310 CLS
00320 X=USR(2057) :REM Get horizontal value
00330 Y=USR(2057,64) :REM Get vertical value
00340 IF X=Z THEN 350 ELSE CURS 66:PRINT [I4 X]
00350 IF Y=T THEN 360 ELSE CURS 80:PRINT [I4 Y]
00360 Z=X : T=Y
00370 GOTO 320
00380 REM #3 : Screen drawing.
00390 HIRES
00400 X=USR (2057):Y=USR (2057,64)
00410 X=8*X REM Expand to screen width
00420 Y=4*Y REM Expand to screen height
00430 SET X,Y : GOTO 400
00440 REM
00450 REM Machine code loader routine.
00460 REM Pokes subroutines to 0800H
00470 FOR A=2048 TO 2077
00480 READ B
00490 POKE A,B
00500 NEXT A
00510 DATA 62,207,211,1,62,128,211,1,201,121,22,32,178,211,0
00520 DATA 219,0,203,127,40,1,170,203,58,48,242,230,3,79,201
00530 REM
00540 X=USR (2048) : REM Initialize the PIO.
00550 RETURN
```

Testing and setting up

Having built the unit, the next step is to type in the test program and check that all's well. It is wise to save the program before plugging in the joystick in case a spike stuffs a bit or two. The first test covers the D-to-A converter and you will need a digital multimeter connected to read the voltage at the comparator's positive input. The most convenient place to attach the probes is across R2. Now run the #1 test and type a few different numbers in, noting that the voltage changes with the number, with '0' you should read around 1.8 volts, while a '63' should read about 2.75 volts. If you go through the range from 0 to 63 you should observe an increase with each number. If you get a decrease at some points then either a 4k7 resistor is open circuit or the data is not getting from the port through to the outputs of the 4050, IC4. ■



Overlay and wiring diagram. Assembly of the pc board and wiring it up are fairly straightforward. Make sure you orient the ICs correctly. Take care in wiring the DB15 plug. Note that pin 8 (0 V) is out of sequence on the ribbon cable. The whole thing mounts nicely in a 'UB1' zippy box, with the joystick mounted through the bottom (which becomes the top), right up one end, leaving enough room to mount the pc board.

PARTS LIST — ETI-674

Resistors ..... all ¼ W or ½ W, 5%
R1,R2,R3 ..... See table
R4-R22 ..... 4k7
R23,R24 ..... 2k7
RV1-RV4 ..... See table

Capacitors
C1,C2,C4 ..... 100n ceramic bypass
C3 ..... 220p disc ceramic

Integrated circuits
IC1,IC2 ..... LM311 etc.
IC3 ..... 4011B
IC4 ..... 4050B

Miscellaneous
Joystick assembly; ETI-674 pc board; zippy box 155 x 95 x 50 mm; DB15P 15-way subminiature plug; ribbon cable; double-sided adhesive tape, hook-up wire, solder, etc.

Price estimate: $32-$35

ALTERNATIVE FOR THE

Now you can have just about any typeface you want on your 'Bee. Just swap the existing EPROM for one that has an 'alternative' character set and unless you're really unlucky you won't need to do any other modifications.

HOW WOULD YOU LIKE a Microbee display that looks like the letters have been cast in hot metal? Or maybe you want some 'space-age' characters that look like they're off the control panel of Dr Who's Tardis. Or perhaps some text 'written' onto the screen with a quill pen.

With other computers you're pretty well stuck with the screen characters they came with, but if you've got a 'Bee you can have

CHARACTER SETS MICROBEE

Tom Moffat
39 Pillinger Drive, Fern Tree
Tas 7101

just about any typeface you want. Why change? To have something a bit different. Variety is the spice of life!

New screen characters can be provided in a temporary fashion, via the Microbee's programmable character generator, or more permanently by the method we will now describe.

Most new Microbees are provided with a facility for telecomputing, that is, using a telephone to dial up some remote computer. This is all very nice if you live in a big capital city where the remote computers live, but if you are in a country area (or Tasmania), every session of telecomputing will most likely cost you a whacking big phone bill as the trunk charges tick over. Eventually we'll be able to telecompute for the cost of a local call, but until then part of your Microbee is going to waste.

Several software suppliers have got onto this idea and are providing program packages in EPROM that can be plugged in instead of the Microbee's 'Networking' or 'Telecom' EPROM. You can always change them back later.

Part of the original telecomputing feature involves re-formatting the 'Bee's screen for 80 columns by 24 lines. This requires smaller characters than normal to fit them all in. The software to do the re-formatting is in the Networking EPROM that is removed to make way for new program packages. But the data for the small character set *stays in the Microbee!* It's in the 'Bee's character generator EPROM, in a part you'll never see unless you attack it in a special way, and that's what this project is all about.

The 'Bee's 'normal' character sets fits nicely in 2K of memory, and the earlier computers had a 2716 EPROM to provide this memory. Newer 'Bees use a 4K EPROM, a 2532. The lower 2K has the normal character set, and the upper 2K stores the 'little' characters.

What we are going to do is swap the existing EPROM for one that has a nice jazzy 'alternative' character set in place of the little characters. If you've got an older 'Bee that only has a 2K EPROM, you'll see how to fix that up as well. But first we have to have some way to get at the new characters without the use of the Networking EPROM.

The Microbee's screen activities are controlled by a 6545 CRT controller chip. The 6545 figures out what character to show by continuously scanning through the screen memory starting at decimal address 61440. But the 6545 has its own address 'hotline' to the actual screen memory chip, so as far as it's concerned the addresses start at 0.

Scanning continues until all the 11 address lines to the screen RAM are cycled, and then it all starts over again. As each new address is fed to the screen RAM, the data that comes out of it is treated as addresses and fed to the character generator, where it selects a group of 16 bytes. Which of the 16 is energised is determined by the 6545 which attacks the character generator directly with four more address lines. Data from each character generator byte comes out broadside where it's applied to an external shift register to be clocked out one bit at a time, as video. Whew!

Now it just happens that the 6545 has 14 main address lines, and only 11 go to the screen RAM. Two don't go anywhere, but the highest order one is tied to the highest address line on the character generator EPROM (in newer 'Bees. On the older ones we'll fix it!). So if we can cause the 6545 to raise that last address line high, it will select the second half instead of the first half of the character EPROM, and produce the alternative character set.

All we have to do to get that last line high is tell the 6545 to start counting from 8192 instead of 0. In BASIC we "POKE 222,32" and then hit Reset. Yahoo! The new characters will be in force until we either "POKE 222,0" or cold start the computer. It is also possible to change character sets as part of a program, as shown in Listing 1. Line 210 gives direct access to the 6545, so you don't have to hit Reset to make the change.

Now some bad news. Because of the way the 6545 generates a cursor, the cursor will not exist when the alternative character set is in use. But what's a cursor between friends? We usually want to get rid of the cursor during the running of a program, and you'll still have a cursor under Wordbee, which generates its own.

Legal modifications

If you're lucky you can bring all these changes about simply by bunging in a new EPROM for the character generator. If you're not so lucky you'll have to do some surgery on the main circuit board. You'll soon find out where you stand if you "POKE 222,32" and then hit Reset. One of three things will happen:

1. Your character set shrinks to little characters. You won't have to touch the circuit board, just put in the new EPROM.

2. The screen goes blank. DON'T PANIC! Use Escape and Reset to cold boot the computer and bring the screen back. Lucky you, the circuit board mod won't have to be done, but you've got a 2K instead of a 4K character generator. Raising the address line just switched the EPROM off altogether. Go ahead and put in your new EPROM.

3. Nothing happens, other than losing the cursor. You will have to do the mod which ▶

```
00100 REM This program displays the standard character set
00110 REM from the EPROM, and then flips back and forth
00120 REM between the standard and alternative character sets.
00130 CLS
00140 FOR I=0 TO 127
00150 POKE 61440+256+I,I
00160 NEXT I
00170 K=32: GOSUB 200: REM sets up alternative characters
00180 K=0: GOSUB 200: REM sets up standard characters
00190 GOTO 170
00200 REM subroutine to select a character set specified by "K"
00210 IN#0 OFF: OUT 12,12: OUT 13,K: IN#0 ON
00220 FOR T=1 TO 1000: NEXT T: REM little time delay
00230 RETURN
```

Listing 1. Shows how to change character sets as part of a program.

```
00100 REM Display the character set currently in the PCG.
00110 CLS
00120 FOR I=128 TO 255
00130 POKE 61440+I,I
00140 NEXT I
00150 END
```

Listing 2. Check your character construction. Run this BASIC program after block-moving your entire work area into the PCG.

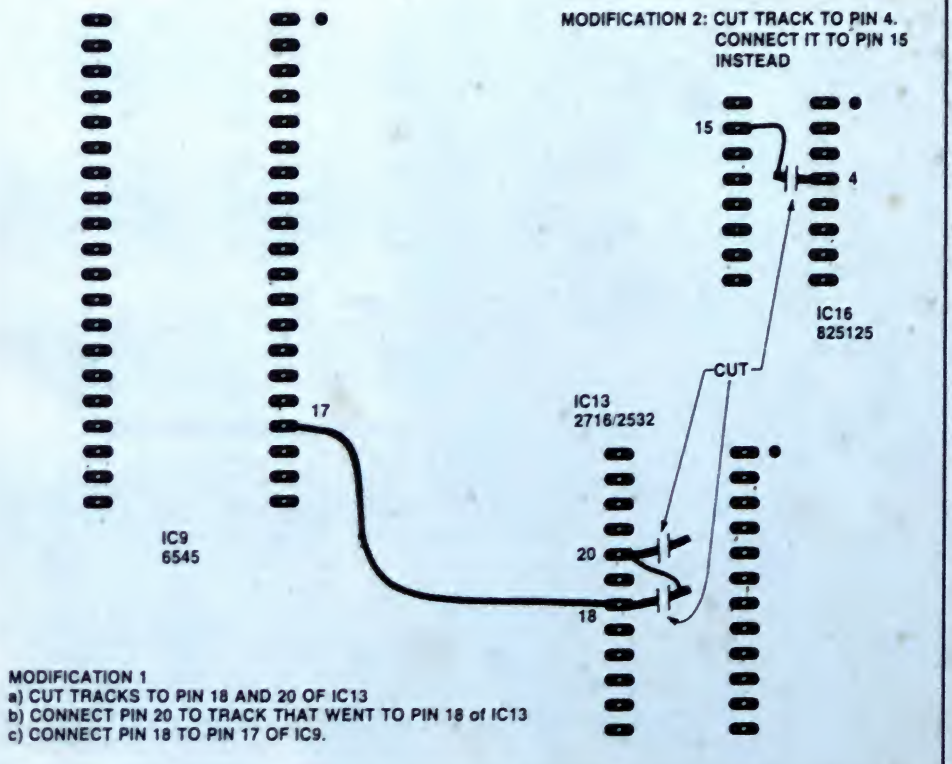


Figure 1. An underneath view of the main Microbee pc board showing the two modifications that may need to be made.

will now be described.

What you are about to undertake is an 'approved Applied Technology upgrade' which explains why many off-the-shelf Microbees already have it done. It must be the first 'approved' project I've tackled; most (like memory upgrades and power supply changes) certainly aren't approved. Being 'legal' somehow takes the fun out of it, but let's get stuck into it anyhow, it won't take long.

Observe Figure 1 which is a view of the bottom of the main Microbee board. You'll have to do some rather fine slicing of tracks and soldering, and if you don't feel up to it any experienced technician should be able to help.

Open the Microbee and remove the memory battery (saved everything?) followed by the top board. You may have to remove some Silastic-type goo to do this. Next remove all the screws holding the main pc board/keyboard assembly into the case. When it comes free the speaker will probably be attached to the case, so don't break any wires. Remove IC13, the character EPROM. Lay the board over so it matches Figure 1 and prepare your scalpel (seriously, a surgeon's scalpel is ideal for this).

Locate IC13's socket from the bottom and carefully cut the tracks at pins 20 and 18. Use a tiny bit of wire to connect pin 20 to the track that used to go to 18. Use a longer bit of wire to connect pin 18 to pin 17 of IC9, the 6545 (this is the new address line). Carefully check your work for shorts and then install your new character EPROM.

Modification 2 has absolutely nothing to do with new character sets; it prevents random flashes from messing up your screen displays. It was included here because it's normally done at the same time as the character EPROM mod, so you might as well knock it over too. And while you're inside the computer perhaps you should consider doing a memory upgrade . . . (oops — shouldn't say that!).

Making new character sets

Figures 2 and 3 show three new Microbee character sets exactly as they appear on the screen. They were worked out on graph paper and then entered directly into memory via the 'monitor', at hexadecimal addresses from 2000 to 27FF. You can purchase them ready to go as detailed below, or make your own.

There are BASIC programs around which let you use the screen instead of graph paper to construct characters, but I've found the 'manual' method easier. Since each character occupies a 16-byte slot in memory, a character's address can be worked out by taking its ASCII value in hex, and making this the middle two digits of the four digit starting address. Thus you would find the letter "A" (hex 41) at address 2410. "B" would be at 2420.

The actual format (which bit goes where) is described in the PCG programming section of the Microbee instructions so we won't repeat it here. You can see how your character construction is going by block-moving your entire work area into the PCG and then running the little BASIC program in Listing 2.

Be sure you occasionally save your work area to tape as well. When you're satisfied with the result you can move your whole character set to 2800, copy the first half of the existing character EPROM to 2000, and then make an EPROM from the data in

2000 to 2FFF. No EPROM programmer? You haven't built the ETI-668 kit? Oh dear. Well hang in there, we'll soon sort that problem out.

The new character sets explained!

Typewriter — I'm using this one in Wordbee as I write, and it looks quite nice. It must be the 'serifs', the little bases some of the letters stand upon. They give a sense of class. I must admit the inspiration for this one came from the Dick Smith Challenger computer.

This type of character is seen in just about all traditional print — newspapers, magazines, even ETI! The little letters in the control characters actually describe what they do — form feed, group separator, etc. The exception is the little arrow for 'return'. This is needed for Wordbee.

Data 70 — This character set most likely originated as Letraset stick-on letters. Letraset is frequently used in laying out advertising, and a lot of people see the results. Now Data 70 pops up in newspapers, on the sides of buildings, on labels on equipment, everywhere. I bet that if you look through this issue of this magazine you'll find Data 70 used somewhere.

A partial version of the character set appeared in ETI several months ago as a BASIC program. This one comes as a data file, and includes all the lower case letters and punctuations.

Script — People do it, enlightened typewriters do it, so why not computers? This character set relies on each character entering and leaving its cell at the same place, so they give the appearance of all being strung together. The numerals aren't of the same character set, they come from an archaic style of writing that sent the characters well above and below the centre line. A program listing in 'script' with those big numbers looks quite silly. But it's all good fun.

Now, we could have published all the hexadecimal code representing these three character sets. It would have resulted in 384 lines of code with 16-bytes in each line, for you to key in under Monitor. Boring to read, and boring to do. Roger Harrison never would have printed it (would he?). So here's what you can do if you want a new face for your 'Bee.

Send $15 to the author for which you will receive all three new character sets on a 1200 baud cassette, in a nice padded bag, postage included. Typewriter will be at 2000-27FF, Data-70 at 2800-2FFF, and Script at 3000-37FF. You can then block move them into the PCG, incorporate them in your programs, put them in EPROM, or whatever.

Alternatively, for $25 I can supply a 2532 type EPROM with the Microbee's standard character set in the normal place, and one of the new character sets in the second half of the EPROM. That's only $5 more than the retail price of a blank EPROM, and it includes postage. The recommended second character set is typewriter, it's really lovely to use, especially with Wordbee. Data-70 looks pretty spectacular, but I fear you would soon tire of it. Script? Well, maybe, but would you use it after the newness wears off?

Remember that this whole process is reversible. If in a few years the cost of telecomputing comes down to a reasonable level, you can always pull out your alternative character EPROM and put the old one back in, and nobody will be the wiser. ■

Tired of boring old "stock-standard" characters on your Microbee? How about making it into a living antique.. a replica of an old time mechanical typewriter. You ought to see this "typewriter" character set go with Wordbee!

At the other end of the time scale is the "Data-70" character set. You'll see this one used in a lot of advertising where a high-tech, "space age" image is desired. In fact "Data-70" has nearly been done to death but it looks good on the Microbee.

This is a demonstration of the "script" character set. I'll bet you've never seen a computer write like this, have you? The numbers look like they'd be more at home on a gravestone... "This computer had its final crash on July 26, 1984. R.I.P."

Figure 2. The three new Microbee character sets exactly as they will appear on the screen. 'Typewriter' on the top, 'Data-70' in the middle and 'Script' on the bottom.

Microbee Standard:

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

Typewriter:

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←`abcdefghijklmnopqrstuvwxyz{|}~

Data 70:

!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←`abcdefghijklmnopqrstuvwxyz{|}~

Script:

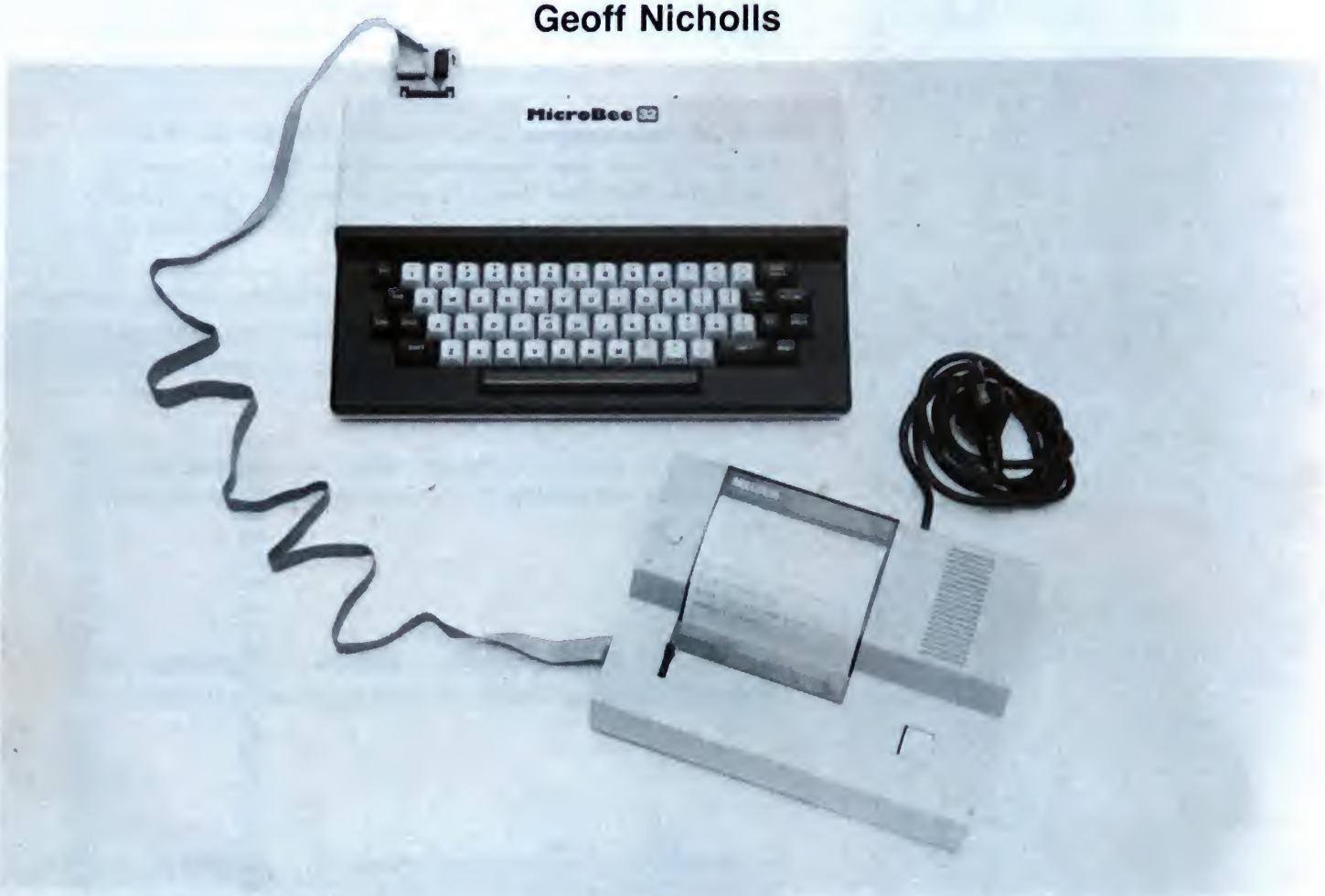
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←`abcdefghijklmnopqrstuvwxyz{|}~

Figure 3. What you see is what you get if you replace the existing EPROM with an EPROM which can generate other type faces.

Parallel printer interface for the Microbee

Nick-named "the 'Bee pee interface", this project allows you to connect your Microbee to a printer having a 'parallel' or 'Centronics' input. It's simple to build and low in cost.

Geoff Nicholls



THERE'S NO SUBSTITUTE for hard copy from a computer in a myriad of situations and applications, but I don't really have to tell you that. There's no doubt that the most cost-effective way to provide hard copy from your Microbee is to buy an old teletype, rig up an interface for it and do some software jiggery-pokery to convert the 'Bee's ASCII output to Baudot. But that's all explained in Project 672 elsewhere in this magazine! However, if your desires and budget stretch to a 'real' printer, then here's how to do it.

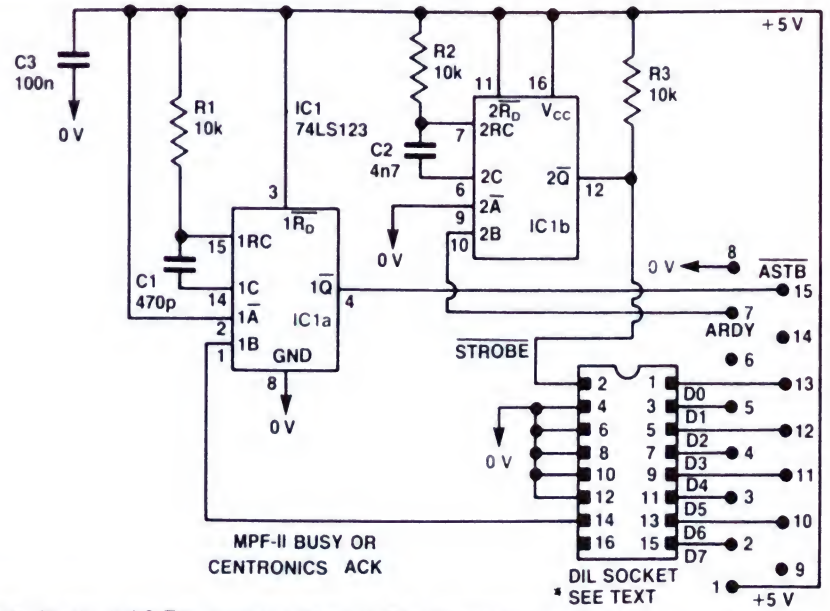
The Microbee's parallel port won't drive a parallel interface printer directly. The 'Bee's parallel port is 'driven' by a Z80 PIO. To signal a peripheral attached to the port that the data's ready and waiting, the PIO puts a 'strobe' signal on pin 2. Trouble is, the STROBE pulse is generally too short. Simple solution? — stretch it! In addition, most parallel interface printers generate an 'acknowledge' signal to let your computer know that they're "busy". The 'Bee's PIO needs an appropriate-length pulse to know that, so I've taken care of that, too.

Next problem is the price of printers. Good quality 80-column dot-matrix printers have recently plummeted in price and can now be obtained for under $500. If that's what you're after, fine, shop around. If, like quite a few readers, you like to (or is that *have to*?) watch your spare cash but really want a dot matrix printer, then finding something gets a little harder.

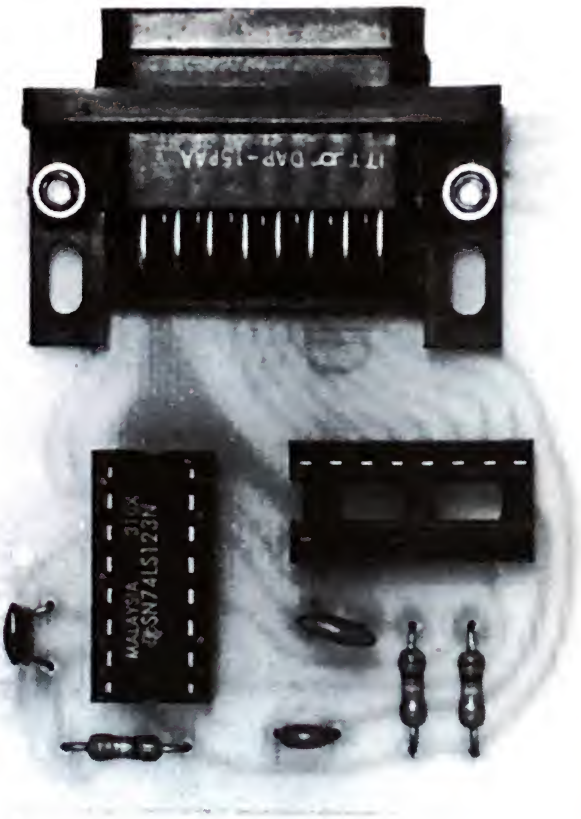
In 'ETI' May we reviewed the Multitech MPF-II computer. With it came a 40-column dot matrix printer. I looked closely at the price and its capabilities. Apart from a comprehensive character set, it included graphics capabilities. Hmmm. At around $230 retail (inc. tax), it represented very good value for money. While 80-column printout is desirable, I found 40-column printout to be no real drawback, especially at that sort of price. The MPF-II Printer was imported and distributed by Emona Enterprises, Suite 204/661 George St, Sydney NSW 2000. (02)212-4815. The company has distributors in all states.

While the construction here specifically relates to this printer, the interface will work equally well with 80-column printers.

(*Note:* The MPF-II printer input circuitry is CMOS and the usual care should be taken to avoid damage.)



Circuit. Simple, huh? The numbers shown in the DIL socket refer to the wire numbers in the MPF II printer ribbon cable, viewed looking down on the DIL socket. The DB15 plug numbering refers to the pins when viewed from the front of the plug (pin 1 uppermost).



HOW IT WORKS ETI-671

The circuit interfaces a Z-80 PIO port to a MPF-II printer. It is also suitable for use with a Centronics style printer.

The timing diagram for the interface is shown in Figure 1.

The PIO ARDY line goes high to signify that a byte is available at the port lines D0-D7 of port A. This triggers Monostable 2 to generate an active low pulse of about 20 μs which signals the printer to accept the byte. The printer will raise the BUSY line while it is processing the byte, or printing a line.

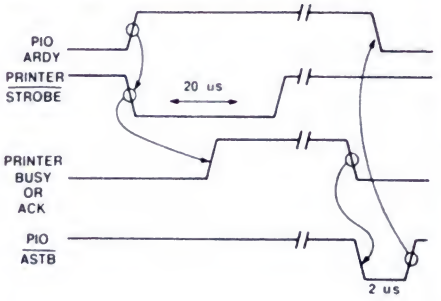
Note that BUSY on the MPF-II is ACKNOWLEDGE for a Centronics printer.

When the printer is ready for more data it lowers the BUSY line thus triggering Monostable 1 and generating an active low pulse of about 2 μs, the rising edge of which terminates the ARDY signal.

These pulse lengths are for the MPF-II. If your printer requires other pulse lengths, change C1 and/or C2 according to the formula

$$t = 4000 \times C$$

where t is in seconds, C is in Farads.



This is only an approximate formula, but should be within 20% or so of the actual pulse length.

PARTS LIST — ETI-671

Resistors ..... all ¼ W, 5% unless noted
R1, R2, R3 ..... 10k

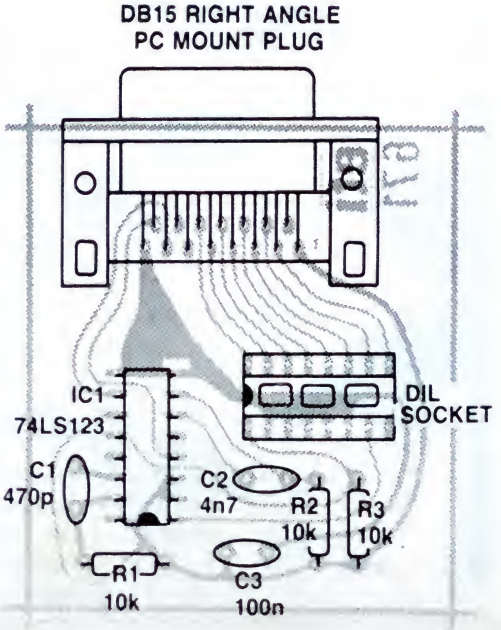
Capacitors
C1 ..... 470p ceramic
C2 ..... 4n7 greencap
C3 ..... 100n ceramic bypass

Semiconductors
IC1 ..... 74LS123

Miscellaneous
ETI-671 pc board; 16-pin IC socket; either an Ansley 609-M165H insulation displacement plug or a 16-pin DIL header; DB15 right angle pc mounting plug; two 6BA ¼" screws and nuts; cable to suit printer.

Estimated cost: $10-$12
(not inc. printer plug & cable)

Below. Full size pc board artwork.



Construction

There's not much to building this one. Start with the right-angle plug, mounting it on the component side as in the photo. It is better to drill the mounting holes and screw the plug to the board before soldering, to relieve stress on the pins.

I used a socket for the IC, and recommend that you do too. The resistors and capacitors are not polarised and may be assembled in any order.

Connecting the printer

If you are using an MPF-II printer, you should use 16-way ribbon cable supplied with the printer. An insulation displacement DIL plug will fit directly onto the ribbon. *Note:* I have utilised a 'reverse' type ID plug, that is, the first conductor in the cable connects to pin 16 of the DIL socket. The plug type I used was an Ansley 609-M165H. Check carefully that the plug you use has the same pinout.

The circuit diagram shows how the cable conductors connect to the DIL socket. The numbers shown refer to the cable, starting at the end with the blue stripe as No. 1. The socket is shown as viewed from above. For example, the No. 1 conductor connects to pin 16 of the socket, the No. 2 conductor connects to pin 1 of the socket, and so on.

If you are using a Centronics printer, you cannot simply attach an ID plug to the Centronics' cable. The most straightforward solution would be to use a DIL header plug, and strip and solder the wires one by one. The accompanying table shows the relation between the DIL pinouts, the MPF-II cable and the Centronics signals.

(*Note:* The MPF-II BUSY signal corresponds to the Centronics ACKNOWLEDGE.) ▶

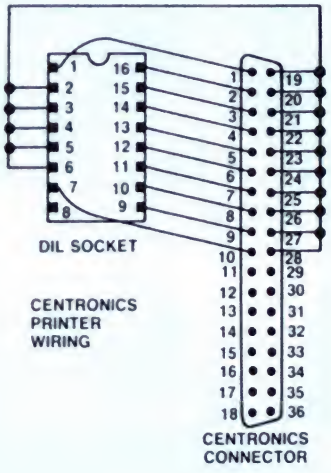
Using the printer

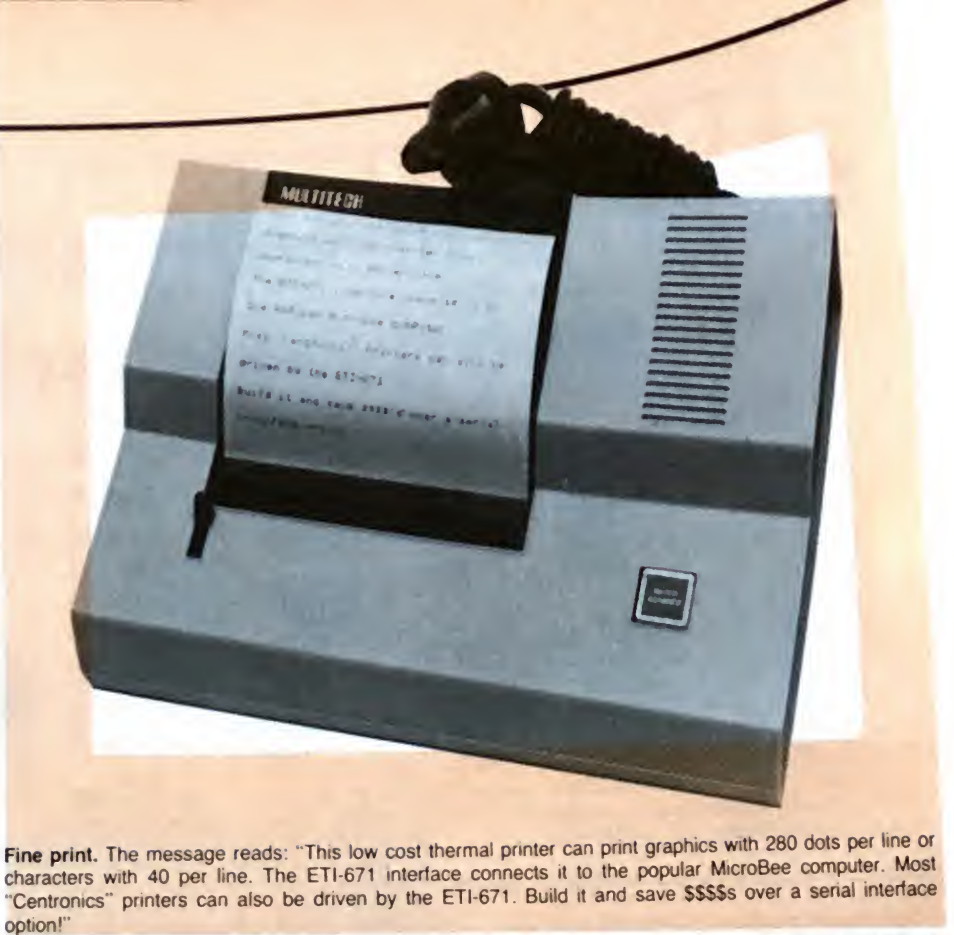
To call the printer from BASIC, it is first necessary to use the output redirection command, to tell the Microbee that a parallel printer is connected. There are two ways to do this. The printer can be the *list* device or the *output* device (or both). To set up the printer as the list device, which means any program LLIST commands will output to the printer, type the following:

OUTL#1.

To make the printer an output device, which means it will print anything that normally appears on the VDU screen, type the following:

OUT#1 ON.



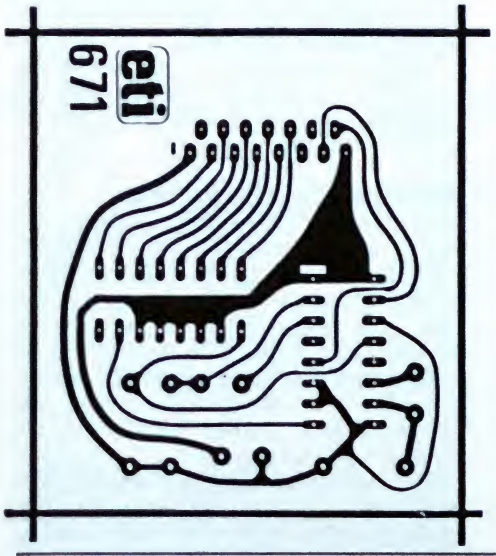


Fine print. The message reads: "This low cost thermal printer can print graphics with 280 dots per line or characters with 40 per line. The ETI-671 interface connects it to the popular MicroBee computer. Most "Centronics" printers can also be driven by the ETI-671. Build it and save $$$$s over a serial interface option!"

The printer will not begin printing until it either receives a carriage return code or fills up the 40-character line buffer. To send a carriage return to the printer when it is an output device, insert the following:

PRINT " " or PRINT CHR(13);.

Most parallel printers automatically carry out a line feed upon receiving a carriage return. The Microbee normally sends both a carriage return and line feed, so double spacing will result in program listing and so on. Applied Technology has supplied a program to overcome this problem; it is reproduced 'elsewhere in this article. The program needs to be run every time a cold start is executed, but may be deleted once it is run. ■



Pin connection table

DIL socket pin Nos	MPF-II printer cable Nos	Centronics connector Nos	Signal name
1	2	1	STROBE
2	4	19,20	GND
3	6	21,22	GND
4	8	23,24	GND
5	10	25,26	GND
6	12	27,28	GND
7	14	10	BUSY(MPF-II), ACK (CENTRONICS)
8	16	N/C	not used
9	15	9	D7
10	13	8	D6
11	11	7	D5
12	9	6	D4
13	7	5	D3
14	5	4	D2
15	3	3	D1
16	1	2	D0

```
00100 REM- Program to Print the MPF-II
00110 REM- standard character set
00120 REM
00130 REM- OutPut device=Parallel Port
00140 OUT#1
00150 FOR N=32 TO 255
00160 PRINT CHR$(N);
00170 NEXT N
00180 PRINT CHR$(13)
00190 REM- OutPut device=VDU
00200 OUT#0
```

```
00100 REM- MicroBee Program to remove
00110 REM- LineFeed character from the
00120 REM- list output stream
00130 DATA 254,10,200,195,248,18
00140 RESTORE 130
00150 FOR I=320 TO 333
00160 READ A : POKE I,A
00170 NEXT I
00180 POKE 180,72:POKE 181,1
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFG
HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmno
pqrstuvwxyz{|}~
```

Print weather maps with your Microbee using our "picture plucker" facsimile decoder

Be the first on your block to have your own weather maps! This project allows you to decode the signals of shortwave stations transmitting 'radio facsimile' weather maps and satellite pictures and then reproduce them on your dot-matrix printer.

Tom Moffat VK7TM
39 Pillinger Drive, Fern Tree, Tasmania 7101

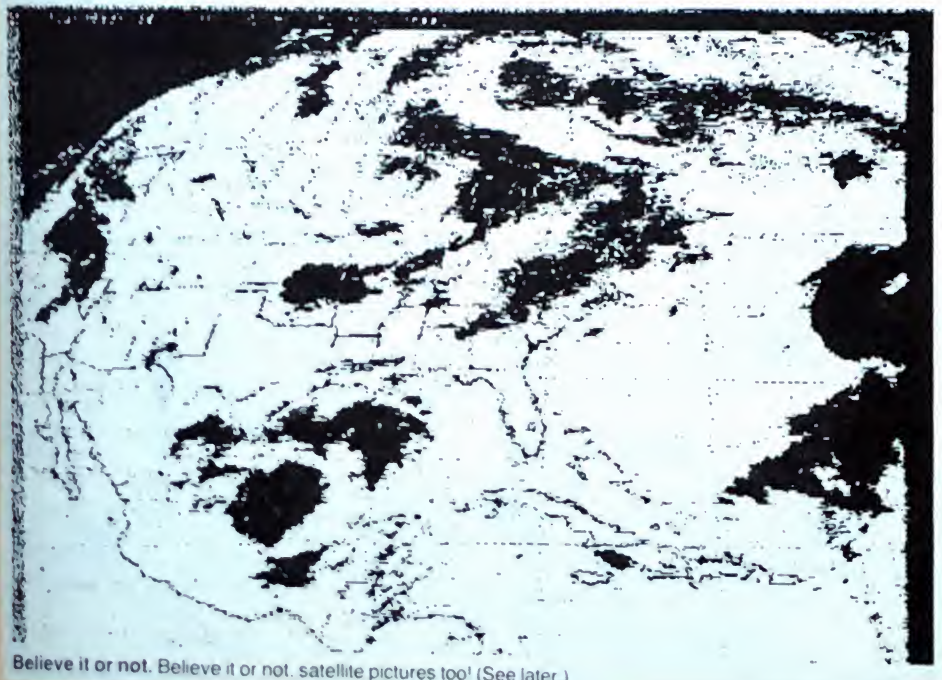
MAY 29, 1983, was the day the drought well and truly broke in Tasmania. The rain poured down from dawn till dusk. The roof leaked and the lawns became mudholes. The farmers rejoiced and everyone else grizzled about their ruined weekend. If they'd had some weather charts they could have grizzled with more authority ... the charts told the whole story in graphic detail!

The accompanying series of charts was received on an ordinary home-style shortwave receiver, a Drake SSR-1 to be exact. The audio was processed by a phase-locked loop decoder and a Microbee computer into graphics print data which was fed to a C-ITOH 8510 printer.

The transmission process is called 'facsimile' which we'll call 'fax' for short. Now I've spelled *that* once, I won't do it any more times. Let's call it *fax*, like the cogniscenti do.

As far as I know this is the first time the world of fax has been opened to the electronics-computing enthusiast. The secret, of course, is the computer.

Traditional fax methods are mostly mechanical, and to home-brew a decent fax machine would require access to a lathe and a lot of time and trouble. They appear to be non-existent on the disposals market. The mechanical fax systems now in use are so well made they seem to last forever. Maybe, when more electronic systems appear ... but we're getting ahead of ourselves. Before we go much further we'd better get down to the bare fax.



Believe it or not. Believe it or not, satellite pictures too! (See later.)

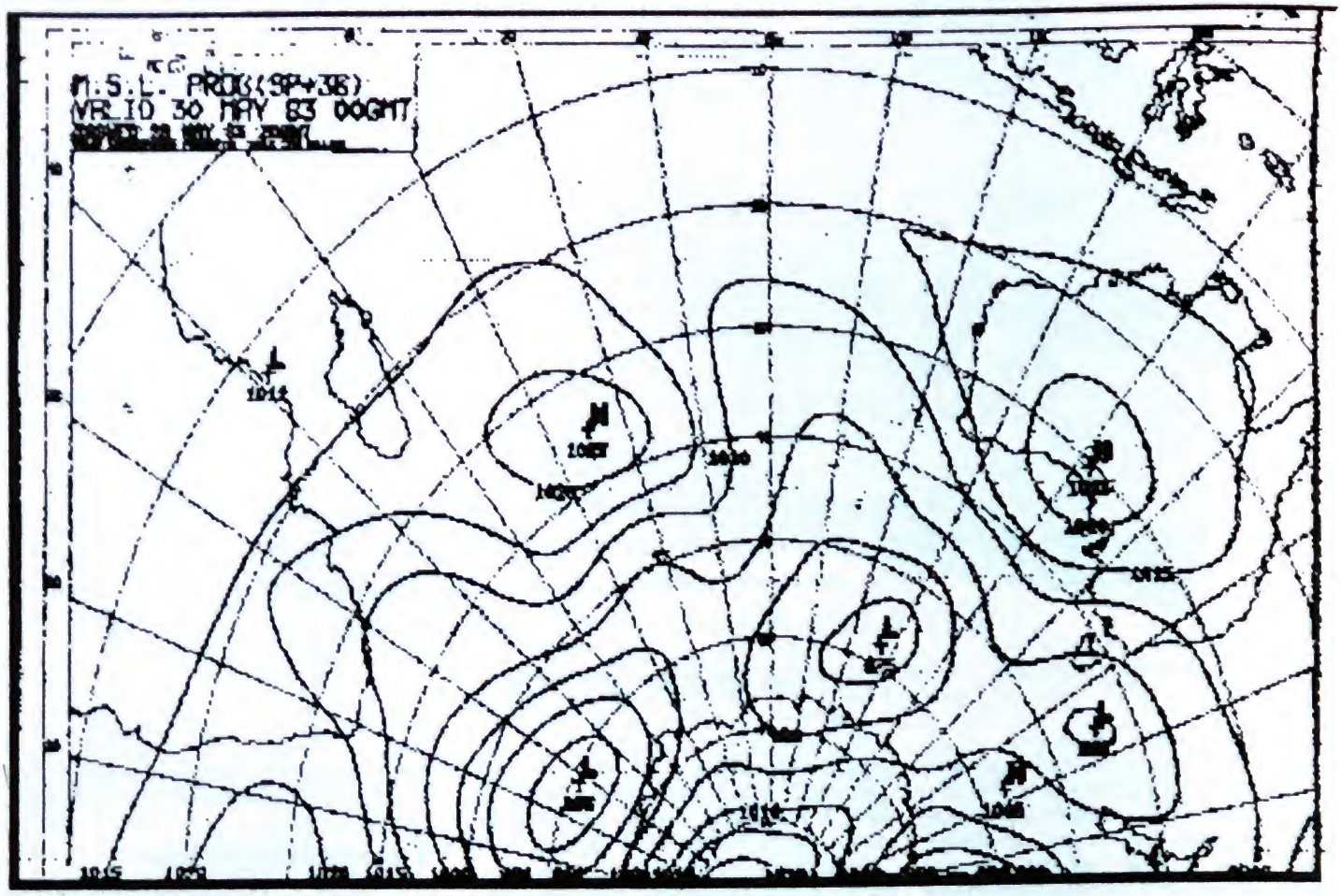
Facsimile explained

The object is to send pictures over a distance. Written data can be transmitted by teletype, but for pictures, or written data in one's own handwriting (such as signatures), fax is the only way to go. Fax is really *veeerry slooowww* television. If the data rate is slow enough you can squeeze an extremely sharp and detailed picture through the restricted bandwidth of a telephone line, or a radio transmitter. Television delivers a complete picture in 1/25th of a second. Fax can take more than 20 minutes for one picture.

Both television and fax work on a system of scanning lines; only the speed of the scanning and the density of the lines is different. Television, with its wide bandwith of 5 MHz or so, is restricted to the line-of-sight coverage of VHF. But since narrowband fax can be transmitted on HF, the range is world-wide, and it makes for some interesting viewing.

The traditional method of fax transmission is shown in a much simplified form in Figure 1. The system consists of two metal drums, one at the transmitter and the other at the receiving end. The two drums are rotating at (hopefully) the same speed. Near the top of the transmitting drum is a lamp illuminating the whole surface.

Above the drum is a 'telescope', a system of lenses, feeding into a light dependent resistor (LDR). The light falling on the LDR is only that from the image of the tiniest pinprick portion of the drum, directly at the focal point of the telescope. The whole telescope is connected to a leadscrew arrangement that moves it slowly along the length of the drum. ▶



The breaking of the drought. Wide-angle view of the southern hemisphere, Africa and Madagascar to the left, Australia to the right. The low that broke the drought has just passed east of Tasmania. Printout shown actual size.

The receiver has a similar telescope over its drum, although there is a lamp in place of the LDR. The receiver lamp is connected via a pair of wires to the transmitter LDR, via a battery to power the lamp.

To the transmitter drum we will attach a piece of paper with some wavy lines drawn on it which is held in place with a piece of black electrician's tape. To the receiver drum we will attach a sheet of photographic printing paper (first turning out the light).

Now we can start the drums rotating in unison. As the telescope at the sending end 'sees' the white paper, the LDR will pass maximum current from the battery via the pair of wires to the lamp at the receiving end. The lamp, through its lenses, will expose at any instant a tiny portion of the photographic paper.

When the first of the wavy lines appears under the sending telescope, its black colour will cause less light to fall on the LDR, less current will pass, less light will fall on the photographic paper, and the image will be reproduced.

As the process continues, lines of whiteness or blackness will begin to build up on the paper, a new line for each rotation of the drum. If the drum turns 1000 times, there will be 1000 lines, quite a high resolution picture, which is better than the television's 625 lines. (Purists will note that since photographic paper is negative-working, the resulting picture will have its blacks and whites reversed. But it illustrates how the system works.) Also note that the image of the black electrical tape will be reproduced. This is important, as we'll see shortly.

Now that we are instant experts on the basic fax system, we can see the problems it generates. If the drums aren't running at exactly the same speed, the picture will slant one way or the other, like on a TV set with misadjusted horizontal hold control.

If the drums don't start rotating from exactly the same angular position, the left of the picture will be on the right and the right will be on the left and the edge will be in the middle. What a muddle!

If the transmitter sends 1000 lines and your receiver needs 1500 lines to cover the paper, your image will be squashed into the upper 2/3 and the lower 1/3 will remain unexposed.

Let's tackle the problems one by one, beginning with the matter of getting the transmitter and receiver to start together. Most fax systems transmit what are called *phasing pulses*; bursts of white against a black background, before the actual picture begins.

One pulse is sent for each line, and the *end* of the pulse, the white to black transition, says, 'the next line starts here!'. At the receiving end the machine adjusts its motor speed faster or slower until the edge of its paper coincides with the finish of each phasing pulse. Normally about 30 seconds of phasing pulses are provided, and if the receiver doesn't 'lock up' in that time the picture is going to be a mess.

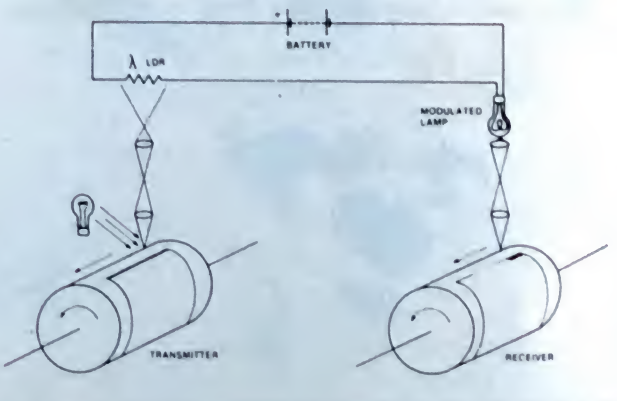
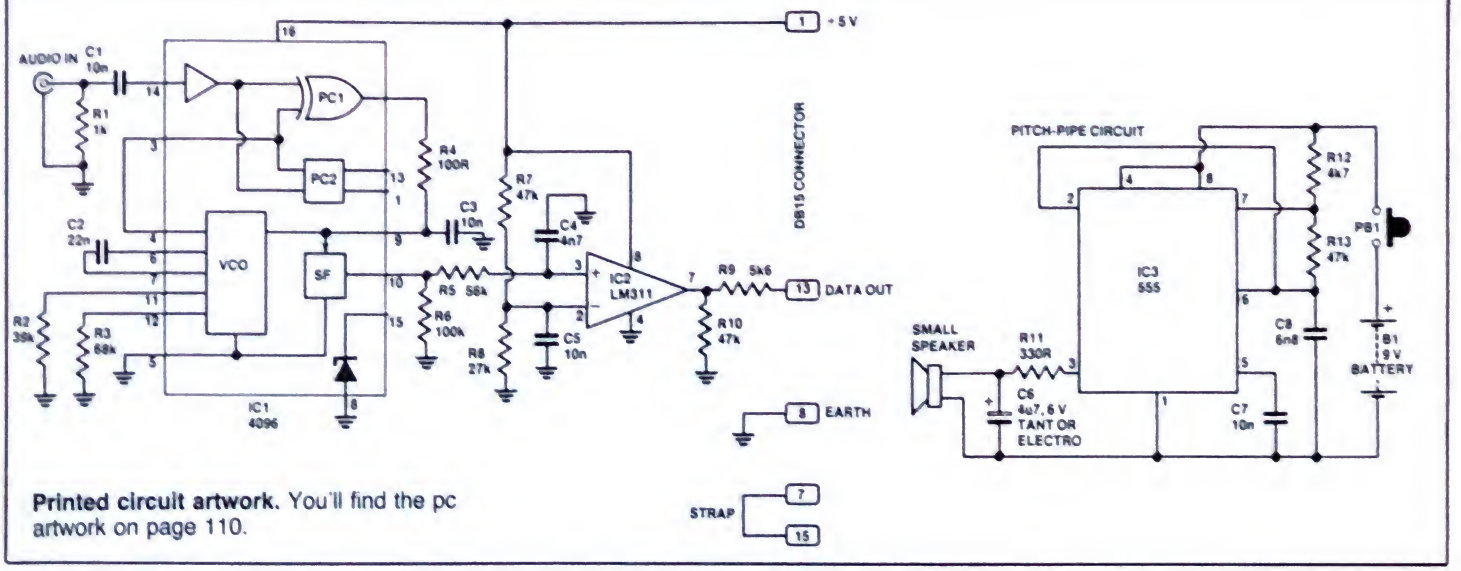


Figure 1. The basic facsimile picture transmission and reception system.



Printed circuit artwork. You'll find the pc artwork on page 110.

Now to the problem of keeping both drums going at the same speed, once phasing has been achieved. There are several ways of going about this, and they're all currently in use.

The simplest method of 'sync' is to use synchronous motors at each end. This method is common in simpler fax machines for use on 'phone lines. It assumes that both transmitter and receiver are running from the same power grid, so their mains supplies are synchronised.

A more elegant way of going about it involves driving both the transmit and receive motors from power supplies derived from crystal oscillators. If the oscillators are stable to within 0.001% good fax pictures will result, and there's no dependence on mains frequencies. So reception is possible aboard a ship, for instance. This is currently the most common sync method.

A third and slightly older system involves the transmission of sync pulses at the start of each line, just like in television. The receiving motor is set to run slightly fast and when it reaches the end of a scan line it slows or stops until told to go ahead by a sync pulse. Sync pulses show up as a black line down the left hand side of a picture (or the right side if it's been sent upside down).

The faithful reproduction of shape is determined by the system's *index of co-operation*. This is the product of the length of a line measured in some unit and the number of lines per the same unit. Confusing? Yup.

Let's try again. Assume that our transmitter drum can take a picture 38.4 cm wide. Assume that it sends 15 scan lines for each vertical centimetre of picture. The system's 'index of co-operation' is $15 \times 38.4 = 576$.

Now assume that our receiving system is only 18 cm wide (which just happens to be the width of our computer graphics printout). If we still went for 15 lines per cm the resulting picture would be very tall and narrow. To stay with the transmitter's index of co-operation of 476, our line density must be 576/18 or 32 lines per cm. If this is so, a circle from the transmitter will look like a circle on the receiver. For various reasons the computer receiver index of co-operation isn't exactly 576, and the resulting circles are slightly 'tall'.

One fact that emerged from researching this article is that there is no single standard for fax transmission. Scan speeds can range from 68 lines per minute (one a second) to 360 lpm. Scan densities can be just about anything.

Radio fax can be sent as AM where the amplitude of the carrier varies with the picture, or as FM where the frequency varies. A picture may run for seven or 10 or 20 minutes and it seems some stations transmit non-stop strips of picture and never any phasing pulses. Pictures may be black and white only (binary), or they may contain many shades of grey (analogue).

From observations on the shortwave bands, certain systems seem popular. Every one I've got printable pictures from seems to use an index of co-operation of 576, which certainly makes life easier. About 80% scan at 120 lpm, about 18% use 60 lpm and the remaining 2% use some weirdo system. All seem to use FM with *white* as the *higher* frequency, and most seem to shift their carriers about 800 Hz between white and black.

So, the computer system described here is set up for the most likely signals: I.O.C. of 576 (or thereabouts), 120 lpm, 'sync pulse' synchronisation, FM with shift in the 800 Hz region, and a picture time of nine minutes which is plenty of time to receive a 'right-way-around' picture from station AXM.

Station AXM

AXM was the inspiration for this whole project. You've possibly heard mention of AXM regarding radio-teletype (ETI, April 1983) which it generally transmits during the first half of every hour. For the rest of the time it sends fax.

I first encountered AXM fax during a trip to the Antarctic last summer aboard the Nella Dan. In the ship's chart-room is a large Japanese-made fax machine that uses a flat bed recorder instead of a rotating drum. Two or three times a day an officer would press a few buttons and the machine ▶

HOW IT WORKS — ETI-736

There are two basic parts to the project: the decoder itself, and the pitchpipe tuning aid. A shortwave receiver, set to receive the upper sideband mode, is tuned to a station transmitting radio facsimile (see main text). The pitchpipe is used to tune the receiver so that the 'high' tone is around 2300 Hz. The receiver output is then recorded on tape. The tape is played back and the decoder inserted between the tape recorder's output and the Microbee's 8-bit port. The computer does the rest.

The decoder is built around a 4046 CMOS phase-locked loop IC. The incoming audio consists of two tones 800 Hz apart. These tones are around 2300 Hz (high) and 1500 Hz (low). They are applied to the PC1 phase comparator input of IC1. The output goes via a low pass filter to the voltage-controlled oscillator (VCO) control input (pin 9). The VCO output goes to the other PC1 input so that the incoming frequency and the VCO frequency are compared. The 'lock' range is determined by R2 and R3. The VCO's free-running frequency is 'pulled' according to the incoming high and low tones.

The 'error' signal generated at the output of PC1 is buffered (after conditioning by the low-pass filter R4-C3) and appears at pin 10. The buffer output at pin 10 is passed to a voltage comparator, IC2, which generates high and low pulses for the input port at pin 13 of the DB15 connector (DATA OUT). Resistors R7-R8 set the non-inverting input of the comparator, IC2, to about 1.8 V. C4-C5 are bypasses.

Supply for the decoder is derived from the Microbee's 8-bit port. Note that the ARDY and ASTB signals are tied at the connector (pins 7 and 15).

The pitchpipe circuit consists of a 555 connected as an astable multivibrator. The oscillation frequency of IC3 is determined by R12-R13 and C8. R11-C6 form a low-pass filter so that a relatively 'pure' tone is issued from the loudspeaker. The output level is low, but then you don't need watts of power to do the job.

would emit from its top a large weather map which was then posted on the wall. As well as serving in more traditional roles, these weather maps would allow the ship's company to estimate how seasick they were going to be the next day.

The fax decoder

To be the first in your block to have 'fax-in-the-home' you're going to need a few items of equipment:

1. A singlesideband shortwave receiver.
2. A good quality cassette recorder.
3. An ETI-736 Fax Decoder and Pitchpipe circuit.
4. A 32K Microbee computer.
5. A C-ITOH 8510 printer (or Epson MX-80).
6. Lots of patience.

I won't launch into a long technical description of the Fax Decoder here. It's a good old phase-locked loop, very similar to the ETI-733 Radio Teletype to Computer Decoder (page 42). This new circuit has been optimised for fax data rates and frequency shifts (both very large) and it is working its poor little guts out just to stay in lock.

Considering what is being asked of it, the decoder works very well and it can recover lockable pictures even in the presence of noise or multipath distortion. With the comparator circuit, the decoder will even decode analogue pictures and turn them into binary along the way; just right for feeding to the Microbee.

The ETI-736 Fax Decoder is made up of two independent circuits, the Fax Decoder and the Pitchpipe tuning aid. If your receiver is in one room and the computer set-up in another, you can separate the two circuits with a hacksaw so the Pitchpipe stays with the receiver.

The software for the Microbee is one of those big bit-shuffling programs that's a natural for machine code. It provides for the collection and storage of precisely 637 200 bits of fax information. That is 1080 lines with 590 dots per line. Actually we compress three adjacent fax lines into one, with each vertical group of three dots logically ORed together.

So, after compression 212 400 bits are stored in 26 550 bytes. They make up 360 lines for the printer. Before a picture starts the program locks onto any one phasing pulse and shows the remainder as a continuous black bar. So it's best to wait till they're nearly finished before running the program. During the run, the program waits for a black sync pulse before it inputs the next fax line.

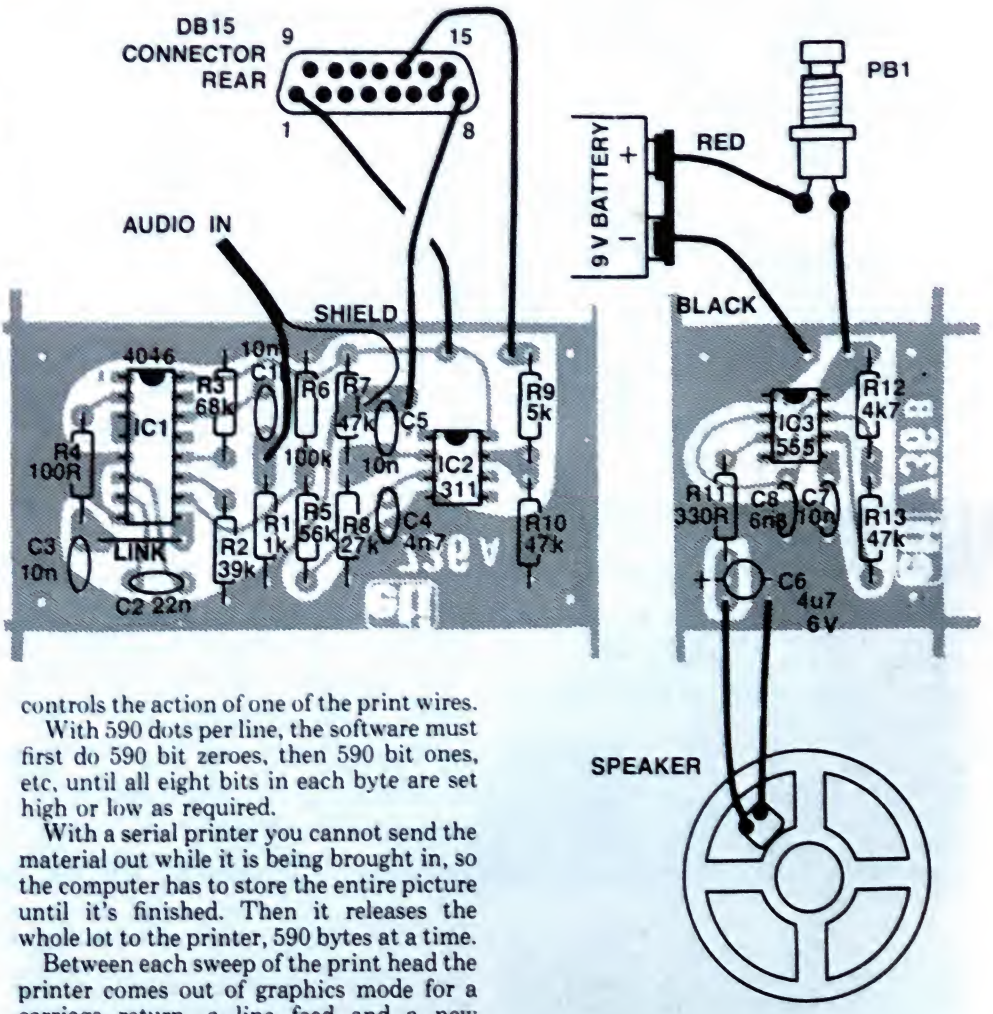
The printer, running in the graphics mode, uses its eight dot-matrix print wires to 'paint' eight lines of fax (24 before compression) onto the paper in one sweep of the print head. A smaller than normal line feed is given so each sweep just touches the one above it. In a byte sent to the printer under the graphics mode, each bit controls the action of one of the print wires.

With 590 dots per line, the software must first do 590 bit zeroes, then 590 bit ones, etc, until all eight bits in each byte are set high or low as required.

With a serial printer you cannot send the material out while it is being brought in, so the computer has to store the entire picture until it's finished. Then it releases the whole lot to the printer, 590 bytes at a time.

Between each sweep of the print head the printer comes out of graphics mode for a carriage return, a line feed and a new command back into the graphics mode.

The DATA items near the end of the program are the commands needed to make the C-ITOH printer do its tricks. The Epson MX-80 uses the same general principle in its graphics mode, so substituting its control codes in the DATA area should make it work the same as the C-ITOH.



Construction

This project is very simple to assemble. First thing to do, no matter whether you've made your own pc board(s) or bought a kit, is to check the board(s) to see that there are no broken tracks, copper bridges (especially between IC pins) and that all holes are correctly drilled. Separate the Pitchpipe board from the Decoder board before assembling the components.

Start construction by installing the resistors and capacitors on the Decoder board. Then install the link at the end of the 4046. Last of all, install the two ICs, making sure you get them the right way round. Now wire the DB15 plug to the board, not forgetting to link pins 7 and 15.

Now tackle the Pitchpipe board. Solder the resistors and capacitors in place first, making sure you put the electrolytic in the correct way. Then solder the 555 in place, taking care to orientate it correctly, also. Finish by wiring in the pushbutton, speaker and battery clip.

PARTS LIST — ETI 736

Resistors ..... all ¼W, 5% unless noted
- R1 ..... 1k
- R2 ..... 39k
- R3 ..... 68k
- R4 ..... 100R
- R5 ..... 56k
- R6 ..... 100k
- R7,10,13 ..... 47k
- R8 ..... 27k
- R9 ..... 5k6
- R11 ..... 330R
- R12 ..... 4k7

Capacitors
- C1,3,5,7 ..... 10n greencap
- C2 ..... 22n greencap
- C4 ..... 4n7 greencap
- C6 ..... 4u7/10 V single-ended electro.
- C8 ..... 6n8

Semiconductors
- IC1 ..... 4046B
- IC2 ..... LM311, uA311
- IC3 ..... uA555, LM555, NE555

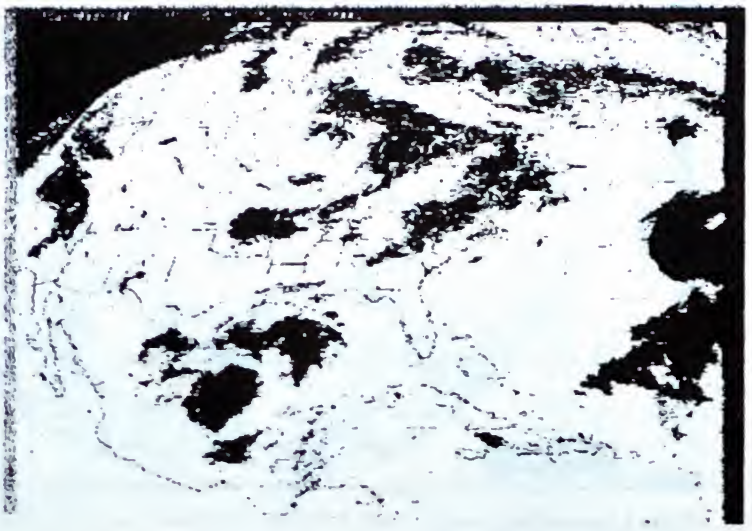
Miscellaneous

PB1 min. ..... momentary-action pushbutton. ETI-736 a and b pc boards; DB15 plug; 50 mm 8 ohm speaker; No. 216 9V battery; jiffy boxes to suit; audio connectors to suit; wire, shielded cable, nuts, bolts etc.

Estimated cost: $25-$30

Picture 3. Satellite picture from AXM, actual size, rotated 90 degrees.

Picture 5. Doctored US satellite picture, transmitted from New York.



If you've got a frequency counter — you only need an audio frequency counter accurate to a few Hertz — then check that the Pitchpipe output is close to 2300 Hz.

Get the picture

Now here's how to collect your pictures: Connect a speaker, nine-volt battery and a press button switch to the pitchpipe circuit. It should beep at 2300 Hz when you hit the button.

With pitchpipe and cassette recorder at hand, tune the receiver until you hear that chorus of crickets called fax; 11 030 kHz is a good place to start. Switch to 'upper sideband'. If a picture is already running, wait for the next one.

The start of a new picture is signalled by a loud *blurt*, actually five seconds of carrier modulated by 300 Hz. Start the recorder. Next, you'll hear the lower tone, punctuated by some 'pip ... pip ... pips' from the higher tone. Push the button on the pitchpipe circuit and tune the receiver until the pips sound the same pitch. This might sound like a rough and ready method of tuning but it's really quite accurate.

Some 30 seconds or so after the pips begin the picture itself should start, with the high tone becoming more dominant. Hold down the pitchpipe button and check the tuning pitch again. Now let the recording continue, checking the tuning from time to time. When the picture is finished the receiver will blurt again; switch off the recorder and take it to the computer.

Now rewind the tape and cue it up so it is sitting just a few pips (phasing pulses) in front of the picture. Set the recorder's volume near where you would to load a tape into the Microbee. With the decoder plugged into the computer's parallel port and the audio line plugged into the cassette 'earphone' jack, roll the tape and then run the program. Turn the printer on and wait. Nine minutes later the computer will beep and your picture will begin painting onto the printer. Two minutes after that you'll be hooked on fax.

About the pictures

All the Australian pictures were received from the Bureau of Meterology's broadcast station, AXM. The 'studios' are located in the Weather Bureau's Melbourne office, although the actual transmitters are in Canberra, operated by the Navy. There's a sister station, AXI, in Darwin.

AXM transmits continuously on four frequencies: 5100, 11 030, 13 920, and 19 690 kHz. Its index of co-operation is 576, scan speed is 120 lpm, frequency shift is 800 Hz and transmission is binary, black and white only, with no shades of grey.

AXM depends on the 'crystal lock' method of sync. It doesn't transmit 'official' sync pulses. But there is something similar to the black electrical tape mentioned earlier ... a black line that always appears on the left hand edge (or the right, if the ▶

Set to go. The completed decoder and pitchpipe, before housing in jiffy boxes.

picture is sent upside down). As far as the Microbee is concerned, it's a sync pulse, and it locks up with ease.

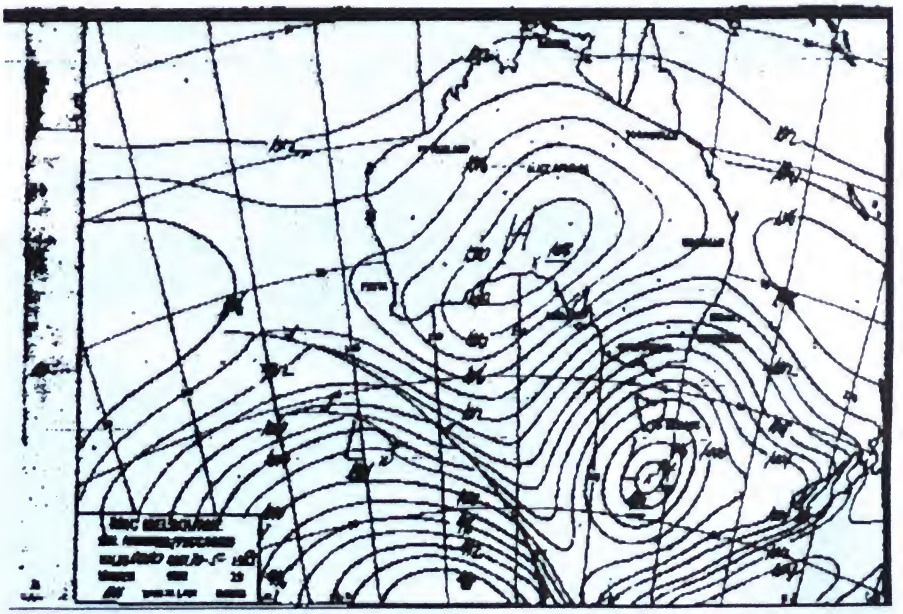
Picture 1 is a standard weather chart just like you see on television every night. Note the big nasty low just south of Tasmania. There's also a cold front sweeping in from the southwest.

Picture 2 is much the same thing, only showing the isobars at 20 000 feet instead of sea level. The low is still much in evidence.

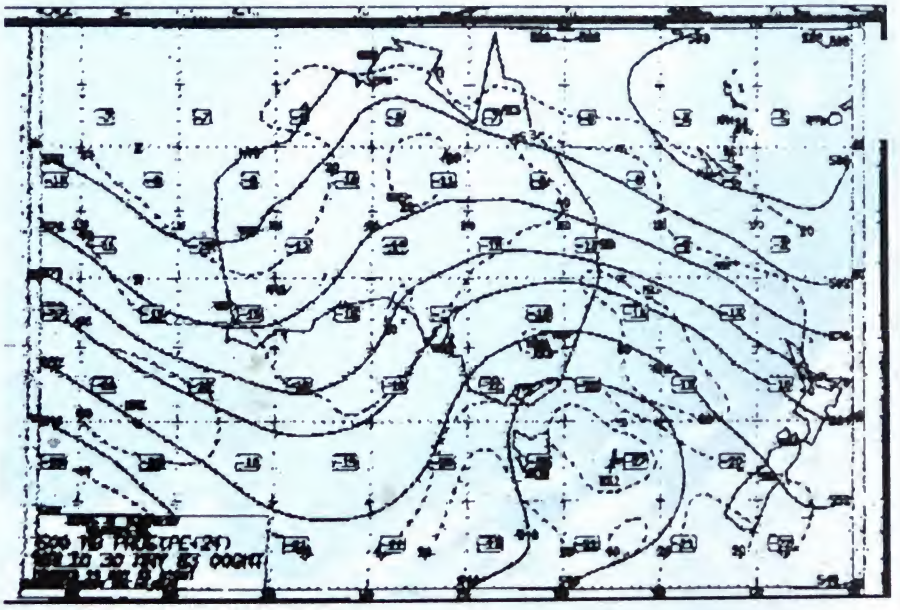
Picture 3 is not a weather map, it's a proper satellite picture transmitted from AXM. Since it's a binary fax system the shades of grey are lost but it's still a pretty smashing view of mother earth, or at least part of it. The picture was too big for all of it to fit in the computer. Rotate it 90 degrees to get the writing the right way up and you'll see the picture in the proper perspective. According to AXM it's the 'southern half of the earth disc as seen from the Japanese satellite GMS'. (That means 'Geostationary Meteorological Satellite.')

Picture 4 is a 'Nephanalysis' . . . a study of the cloud situation in the southern hemisphere. The raw data is a satellite picture which is interpeted and re-drawn by meteorologists, with special symbols indicating the cloud types. Again, that big low blots out Tasmania. Obviously this type of chart would not have been possible before the days of satellites.

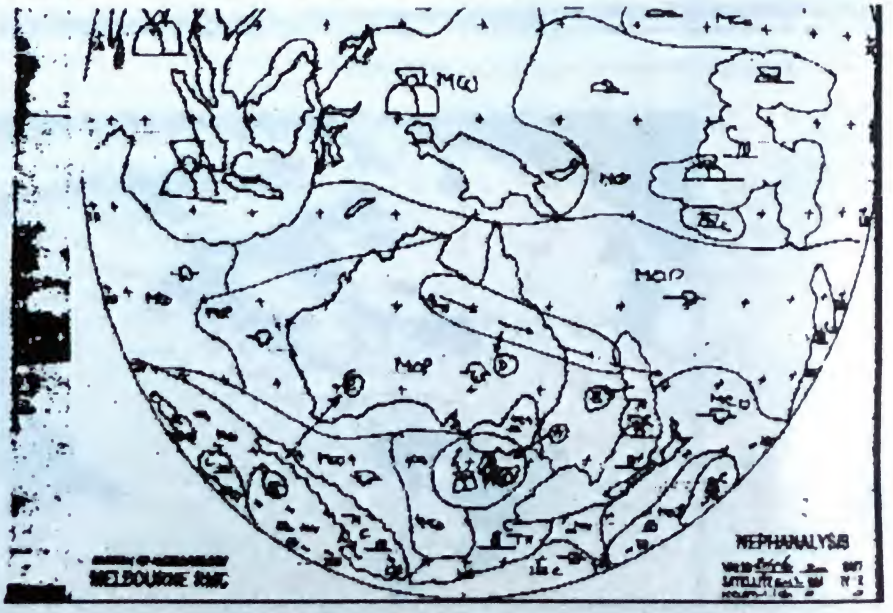
Now to **Picture 5**. You can say you're amazed and astounded now, or later. This one originated from one of America's NOAA satellites and was transmitted from space to the USA where it was souped up



Picture 1. A standard weather chart, from station AXM, 11:30 am, 29 May '83.



Picture 2. Similar to Picture 1, but isobars at 20 000 ft; 2.30 pm same date.

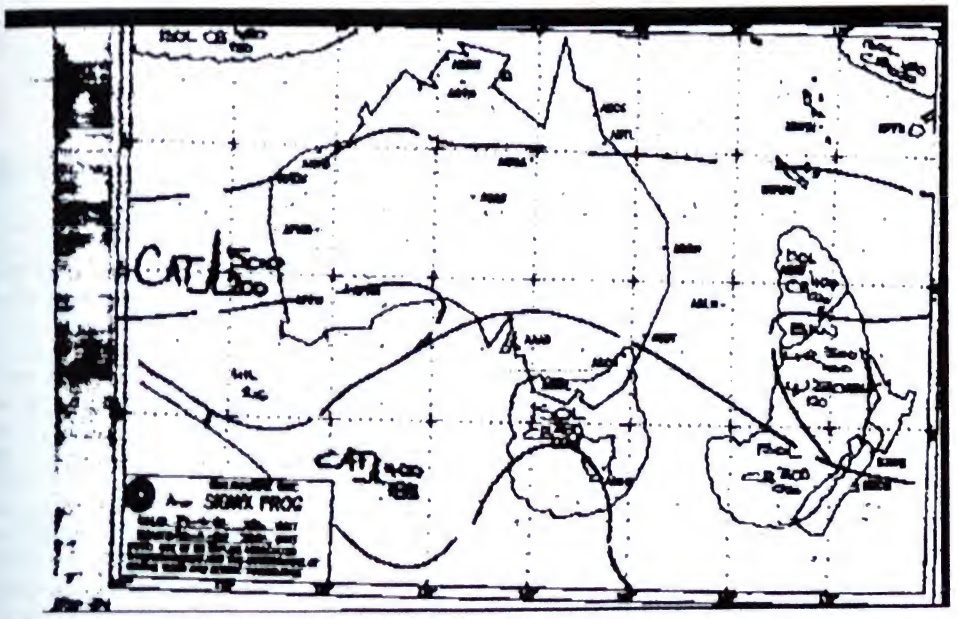


Picture 4. Nephanalysis, 11.45 am 29 May '83, AXM 11 030 kHz.

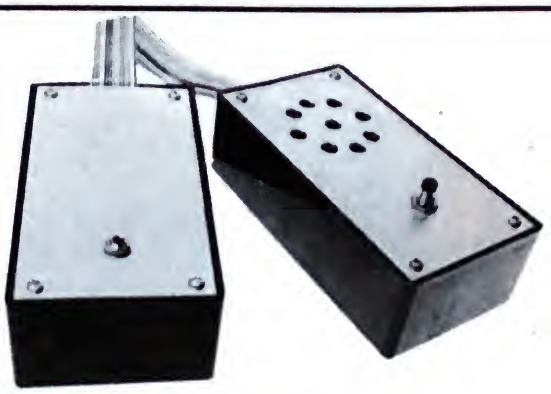
The rest of Australia is gloriously clear! with state borders and the like. Then it was sent back out on HF radio many thousand more kilometres into my humble cassette recorder. In other words it's 'fax DX'. It is an analogue picture that was forced into binary state by the decoder's comparator. It has also had its video inverted in the computer.

Picture 6 is a 'Significant Weather Prognosis', designed especially for pilots. It shows high altitude wind trends and jet streams, and significant cloud areas. The word 'CAT' means 'clear air turbulance' to be avoided if at all possible. This particular example looks like it was put out by the Queensland tourist bureau. It shows Tasmania smothered in cloud (that low again), and New Zealand is about to cop it as well.

Those 'mostly black' satellite pictures, like Picture 3, play merry hell with printer ribbons, so we've turned Picture 8 into 'mostly white'. It still looks OK, huh? That is, if you like black clouds. To invert the video on any picture change the data at 0426 (hex) from 38 to 30.



Picture 6. One for the pilots, another AXM picture.



Housing. Two small jiffy boxes served admirably to house the decoder (left) and pitchpipe (right). The ribbon cable goes to the DB15 plug which plugs into the Microbee user port.

So now you know all about fax, as do thousands of other enthusiasts throughout Australia. Perhaps home fax will join home video and compact disc and all the latest electronic pastimes.

Maybe AXM will even join the ratings race, "Hey folks! You're on AXM! 5000 watts of picture power! And tonight a request from Melbourne listener Randy Oldfellow! Randy asks us to *drop* tonight's *nephanalysis* and play a picture of Bo Derek instead. OK, Randy, this one's for *you!*"

Seriously, though, fax can be useful, and it's certainly a lot of fun. Faxinating. I'd like to thank all those people at the Bureau of Meteorology who suffered my silly questions and helped with the preparation of this article. ●

PROGRAM LISTING

```
ADDR   CODE     LINE   LABEL  MNEM    OPERAND

                00100 ;Facsimile Receiver Program by Tom Moffat: May 20, 1983
                00110
0400            00120          DEFR    16
0400            00130          ORG     0400
                00140
                00150 ;Clear the data store:
                00160
0400 210010     00170          LD      HL,1000 ;START OF PICTURE STORE
0403 110110     00180          LD      DE,1001
0406 01B667     00190          LD      BC,67B6 ;CLEAR 26550 BYTES
0409 3600       00200          LD      (HL),0
040B E5         00210          PUSH    HL
040C EDB0       00220          LDIR
                00230
                00240 ;Synchronize the computer with a phasing pulse:
                00250
040E CD7504     00260 LP1      CALL    SAMPLE
0411 30FB       00270          JR      NC,LP1  ;WAIT FOR WHITE
0413 CD7504     00280 LP2      CALL    SAMPLE
0416 38FB       00290          JR      C,LP2   ;WAIT FOR BLACK
                00300
                00310 ;Start of picture:
                00320
0418 E1         00330          POP     HL      ;POINT TO START OF STORE
0419 114E02     00340          LD      DE,24EH ;590 DOTS PER LINE
041C 01012D     00350          LD      BC,2D01 ;LINE COUNTER AND ROTATING BIT
041F C5         00360 LP4      PUSH    BC
0420 0603       00370          LD      B,3     ;3 FAX LINES = 1 PRINTER LINE
0422 CD7504     00380 LP3      CALL    SAMPLE
0425 7E         00390          LD      A,(HL)
0426 3001       00400          JR      C,$+3   ;SKIP NEXT IF WHITE
0428 B1         00410          OR      C       ;PRINT A DOT
0429 77         00420          LD      (HL),A
042A 23         00430          INC     HL
042B 1B         00440          DEC     DE
042C 7A         00450          LD      A,D
042D B3         00460          OR      E
042E 20F2       00470          JR      NZ,LP3  ;FOR ONE COMPLETE FAX LINE
0430 DB00       00480 SYNC     IN      A,(0)
0432 0F         00490          RRCA
0433 30FB       00500          JR      C,SYNC  ;WAIT FOR BLACK
0435 114E02     00510          LD      DE,24EH ;FOR 590 HORIZONTAL DOTS
0438 ED52       00520          SBC     HL,DE
043A 10E4       00530          DJNZ    LP3
043C C1         00540          POP     BC
043D CB01       00550          RLC     C       ;ROTATE THE BIT
043F 30DE       00560          JR      NC,LP4  ;GO TO START OF LINE
0441 19         00570          ADD     HL,DE
0442 10DB       00580          DJNZ    LP4
0444 0607       00590          LD      B,7
0446 CD0CB0     00600          CALL    B00CH   ;BEEP WHEN FINISHED
                00610
                00620 ;Send out the picture to the printer:
                00630
0449 218904     00640          LD      HL,DATA1
044C 110600     00650          LD      DE,6
044F CD7E04     00660          CALL    PRINT   ;SET LINE FEED PITCH
0452 062D       00670          LD      B,2DH   ;45 GROUPS OF 8 LINES
0454 210010     00680          LD      HL,1000 ;START OF DATA STORE
0457 E5         00690 LP5      PUSH    HL
0458 21BF04     00700          LD      HL,DATA2
045B 1E08       00710          LD      E,8
045D CD7E04     00720          CALL    PRINT   ;CR/LF AND GRAPHICS INIT.
0460 E1         00730          POP     HL
0461 114E02     00740          LD      DE,24EH ;590 DOTS PER LINE
0464 CD7E04     00750          CALL    PRINT   ;8 LINES OF PICTURE
0467 10EE       00760          DJNZ    LP5     ;IF NOT 45 GROUPS
0469 21BF04     00770          LD      HL,DATA2
046C 1E02       00780          LD      E,2
046E CD7E04     00790          CALL    PRINT   ;CRLF
0471 2AA200     00800          LD      HL,(0A2);PICTURE FINISHED,
0474 E9         00810          JP      (HL)    ;JUMP TO MONITOR
                00820
0475 3E63       00830 SAMPLE   LD      A,63    ;SET DELAY TIME BETWEEN DOTS
0477 3D         00840          DEC     A       ;847 uS PER DOT
0478 20FD       00850          JR      NZ,$-1
047A DB00       00860          IN      A,(0)   ;GET A PIXEL
047C 0F         00870          RRCA            ;SHIFT IT TO CARRY
047D C9         00880          RET
                00890
047E 7E         00900 PRINT    LD      A,(HL)
047F CD7FA8     00910          CALL    0A87FH  ;1200 bd SERIAL PRINT ROUTINE
0482 23         00920          INC     HL      ;GET NEXT BYTE
0483 1B         00930          DEC     DE      ;BYTE COUNTER
0484 7A         00940          LD      A,D
0485 B3         00950          OR      E
0486 20F6       00960          JR      NZ,PRINT
0488 C9         00970          RET
                00980
0489 1B54       00990 DATA1    DEFW    541BH   ;ESC T
048B 3136       01000          DEFW    3631    ;1 6    ;LINE FEED PITCH
048D 1B58       01010          DEFW    581BH   ;ESC I  ;INCREMENTAL MODE
048F 0D0A       01020 DATA2    DEFW    0A0DH   ;CR LF
0491 1B53       01030          DEFW    531BH   ;ESC S
0493 3035       01040          DEFW    3530    ;0 5
0495 3930       01050          DEFW    3039    ;9 0
0000            01060          END
00000 Total errors

DATA2   048F    LP5     0457    PRINT   047E    DATA1   0489
SYNC    0430    LP3     0422    LP4     041F    LP2     0413
SAMPLE  0475    LP1     040E
```

Radioteletype-computer decoder

This simple project allows you to hook up your MicroBee to a receiver and print radioteletype messages on the VDU screen. A simple bit of software does the decoding. The project can also be adapted for use on other Z80-based systems.

Tom Moffat VK7TM

IF YOU OWN a MicroBee, or other Z80-based computer, you can now set up your own teletype listening post. But you can forget the old bucket of bolts teleprinter, the computer now serves that purpose.

Besides the computer itself, you'll need to write in the 'Bees parallel input port, build the new ETI-733 RTTY decoder and have a decent HF receiver capable of copying single sideband signals.

You may remember, back in August 1979, a project called the ETI-730 Radioteletype Converter. That design worked well then and it works well now. It's still quite current and it appears that hundreds of them have been built around Australia. This new design is meant to complement the ETI-730. Under rough reception conditions, the '730 will win every time, but under reasonable signal conditions the ETI-733 comes out ahead in ease of operation, speed range and general usefulness. Perhaps a head-to-head comparison is in order; see Table 1.

The performance differences are due to completely different design concepts. The purpose of a decoder is to turn a varying audio tone into a dc voltage proportional to the incoming tone frequency. In the case of radioteletype there are only two frequencies representing two dc levels . . . 'mark' and 'space'. The tones are separated by an amount known as the 'shift' (see the accompanying panel).

The ETI-730 uses filters to recover the two tones from whatever other rubbish may be coming out of the receiver. It compares the level of the tones and whichever one is stronger gets the nod from the logic circuitry. There is another filter in the logic area that discourages transitions faster than 50 per second. The effect of all this is to allow the copy of signals that are sometimes even too weak to hear.

	ETI-730	ETI-733
Ease of Construction	Moderate	Easy
Cost	Moderate	Cheap
Ease of Operation	Fiddly	Easy
Weak Signal Performance	Good	Fair
With Interfering Signal	Good	Poor
High Data Speeds	Poor	Good
Decode Analogue Signals	No	Yes

Table 1. Comparing the 730 and 733.

Gasp! . . . a PLL!

The ETI-733 is based on a phase-locked loop. This will bring screams of anguish from RTTY purists. They'll tell you phase-locked loops are no damn good on HF signals and only marginally useful on VHF. I must admit I experimented with an NE565 PLL chip during the design of the ETI-730 converter and I found results were hopeless. But now, some three years later, from the depths of a CMOS logic data book, comes the *4046 Micropower Phase-locked Loop*, and it goes like a ripper. Why this one should work when the 565 didn't, I can't explain, but work it certainly does.

You've probably heard of PLLs as part of frequency synthesisers. When the cost of crystals shot out of sight, PLLs became a necessity in multi-channel transceivers. The basic PLL has a voltage controlled oscillator and a phase comparator. The signal to be decoded is fed into the phase comparator along with the output of the VCO. The comparator generates an error voltage that's fed back to the VCO (that's the loop part). The VCO then adjusts its frequency to match that of the incoming signal. So the VCO output is a cleaned-up carbon copy of the signal from the receiver and the error voltage is our recovered data signal.

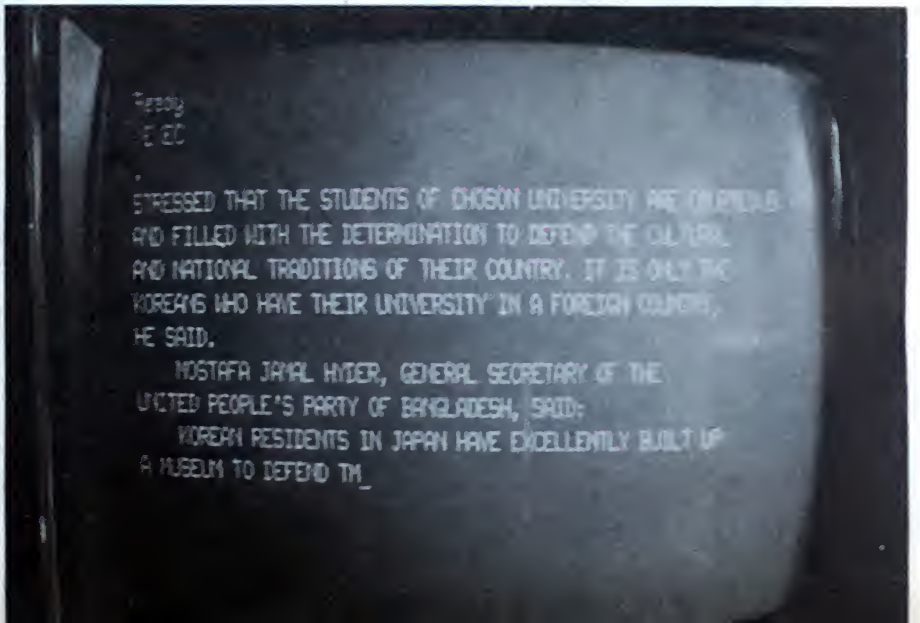
When everything is hanging on and the VCO is following the input signal, the PLL is said to be 'in lock'. If the incoming signal is too high or too low in frequency, beyond the range where the VCO can be pushed to match it, the whole procedure falls apart and the PLL 'loses lock'.

One of the advantages of the PLL system is that you can put a low pass filter in the line between the phase comparator and the VCO which effectively averages out higher frequency noise on the incoming signal, allowing the loop to remain in lock. A disadvantage is that it will always lock onto the strongest signal within its locking range, even if it's not the signal you're trying to receive.

The 4046 chip has a few extra goodies. One is a second phase comparator working on a complicated 'digital' principle. There's also a source-follower to buffer the loop signal for output, and even an on-chip zener diode for power supply regulation. In this design we've ignored the zener diode and the digital phase comparator. The latter was thoroughly tested but the 'normal' phase comparator worked better in this application.

Overall design

Circuit constants were found strictly by the eclectic empiricist method (i.e: trial and error!). After starting with the data book's suggested values, a tape of rather scruffy off-air RTTY signals was played into the



Your 'glass teletype' terminal. Add a general coverage receiver, the '733 decoder and a MicroBee (or similar Z80 system)!

PLL and the results sent to the mechanical teleprinter. A count of errors was made, a circuit constant changed, and then the same bit of tape was played again to see if the errors got worse or better.

It was a slow business but I managed to zero in on what appears to be the best performing circuit.

The lock range is from 1400 to 3400 Hz, in the higher part of the receiver's audio passband, and the loop filter constants are such that signals beyond 300 baud can be recovered.

As well as the PLL chip, the ETI-733 decoder uses an LM324 quad op-amp for input and output conditioning. One section raises the audio level from the 'recorder' output of the receiver to a level more suitable for the PLL.

The PLL output goes to two inputs of another op-amp used as a comparator. One input gets the PLL signal, lightly filtered. The other also gets the PLL signal, but this time via a very long time constant filter to produce an average of the PLL swings. This line is buffered by an op-amp, which also drives a tuning meter.

The arrangement causes the comparator to make a firm decision as to whether a mark or a space is being received, while allowing the signal to drift all around within the PLL's lock range. Shift selection is no longer required . . . anything that crosses the comparator threshold is considered valid, and signals down to 170 Hz shift work nicely.

Since the PLL lock range is 2000 Hz, a lot of receiver drift is tolerable before copy is lost.

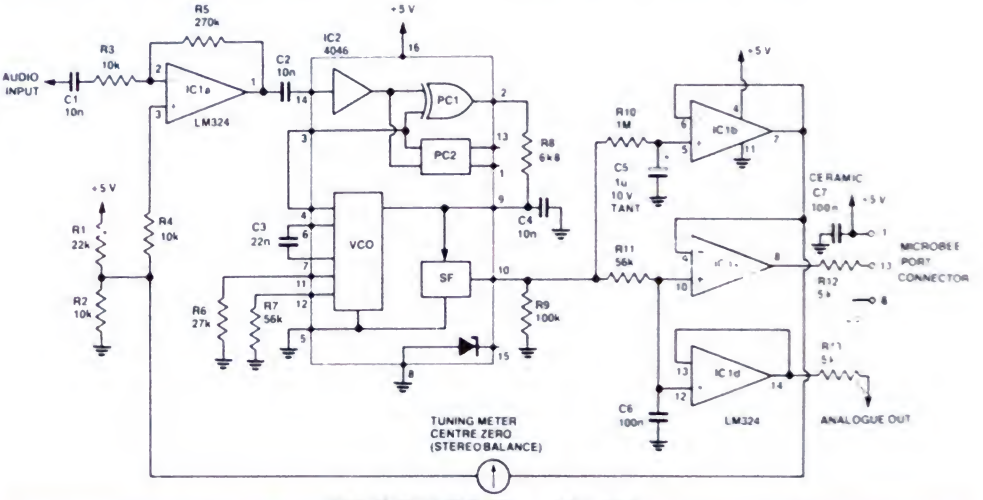
There is one disadvantage with this averaging system . . . teletype signals sent slowly, by hand, will be hard to copy. This is because the signal is spending most of its time on 'mark' and the comparator will drift toward mark, losing its centre reference. But that's of little worry, most interesting signals are sent by machine anyway.

Another op-amp enables a feature that has been examined by oscilloscope, but not properly tried yet . . . an analogue output. The PLL can track anything, not just two discrete audio tones. This opens the door for signals using frequency modulation, such as satellite pictures, weather maps, any facsimile-type signals — and these abound on the HF bands.

When one of these is being received an oscilloscope shows a nice video signal from the analogue output. A signal restricted to peak black and peak white appears on the digital output. A bit of playing around with a computer should result in a method of displaying analogue material on the screen. That's certainly one to work on.

Once the digital signal is available at the decoder's output, you can send it one of two places: to a teleprinter, or to a computer. For the teletype option you can pinch the loop driver circuit from the ETI-730 article.

(See ETI, August '79, page 43. Delete the two resistors, R29 and R30, and LED1 from the emitter circuit of Q2 and take the emitter directly to earth. Feed the PLL decoder output to the base of Q2 via R27.)



HOW IT WORKS — ETI-733

The heart of the decoder is IC2, a 4046 'micropower phase-locked loop' IC. The incoming signal from the receiver consists of two audio tones — one to represent the 'mark' signal, the other to represent the 'space' signal. The 4046 converts these to a two-level digital signal, which is filtered, then buffered and sent to the computer.

IC1a and associated components forms an ac amplifier with a rolloff below 1.5 kHz (i.e: high pass) and a passband gain of about 27. Resistors R1 and R2 form a voltage divider providing a reference level for IC1a of about 1.6 V.

The amplified signal from IC1a drives the signal input of the PLL, IC2, via capacitor C2, which enables the self-biasing input amplifier of IC2 to work with weak signals.

The PLL phase capacitor, PC1, output is filtered by R8 and C4 to obtain an error signal for the PLL's VCO which should oscillate near the incoming frequency present at any particular time. The VCO frequency is 'pulled' according to the actual mark and space frequencies of the incoming RTTY signal.

The free-running frequency of the VCO is about 2.4 kHz, set by C3 and R6. The lock range of the VCO is determined by R6/R7 and is about 1.4 — 3.4 kHz.

The error signal out of PC1 of IC2 is buffered and appears at pin 10 of the PLL. A low pass filter, formed by R10, C5 and IC1b, removes the fast-changing data components of the error signal, leaving a nominally-dc component at the output of IC1b.

The PLL error signal is also fed to IC1c via another low pass filter formed by R11 and C6. IC1c is connected as a comparator with a reference signal on its inverting input set by the nominal dc level from the output of IC1b. This comparator 'extracts' the data from the error signal. This scheme relies on the input not containing long periods of only one input tone, which is usually true.

The tuning meter connects between the 1.6 V reference provided by R1/R2 and the output of IC1b. This indicates when the receiver is producing audio tones shifting within the PLL lock range. When correctly tuned, the current through the meter is zero, hence the necessity of a centre-zero meter.

The data low pass filter, R11/C6, has a roll-off around 30 Hz, which ensures clean data out. However, some experimentation can be carried out with different data rate signals. You can vary C6, decreasing its value for high data rates, settling on a value which gives best results.

Software

The program that follows was written for the MicroBee, but since it's in machine code it should work on just about any Z80-based machine with only minor modifications to some addresses. Here's how it works.

The aforementioned in/out port is first set up in what's called the 'control' mode . . . you specify some bits as inputs and others as outputs, and no 'handshaking' signals are required. In this case I've called all the bits inputs, although only bit 0 is used to bring in the teletype signals decoded by the ETI-733.

A teletype signal is made up of a start pulse, five data pulses, and an extra-long stop pulse (start and stop refer to shaft rotation that takes place in a mechanical teleprinter). Refer to Figure 1, below. A pulse, in the case of 50 bauds, is 20 ms long.

The program first looks for a start pulse which unleashes the following series of events: Bit 3 of an 8-bit register (register C in the Z80) is set high. Thirty milliseconds later the transmitting teletype should be in the centre of its first data pulse . . . this is read and pushed into the right end of the C register. Everything already in C is shoved to the left to make room. Twenty milliseconds later comes the next data pulse, this is loaded into C and everything else shifts along. After five data pulses the bit originally set in C falls out the left hand end, telling the program that a character is finished. Register C now contains, in its first five bits, a binary number between 0 and 31.

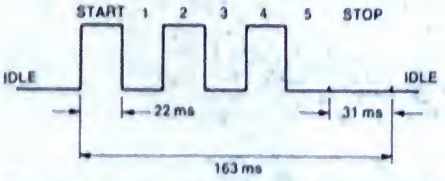


Figure 1. How the teletype signal is made up. For 45.45 baud speed standard, the first five pulses are 22 ms long, followed by a 31 ms 'stop' pulse giving 163 ms per character, while a 50 baud system uses 20 ms pulses with a 30 ms 'stop' pulse, giving 130 ms per character.

In the next part of the program the character in C is inspected to see if it's a figures or letters shift. If so, a flag (register E), is set accordingly. Now register pair HL is set to point to a table of ASCII characters. The number in C is doubled and then added to HL, making HL point to a pair of table entries. Which entry of the pair gets selected depends on flag E which, if zero, gets even numbered entries (letters characters). If E is one, HL selects odd numbered entries (figures and punctuations).

The ASCII character eventually selected is loaded into register B and then sent off to the MicroBee's VDU routine, where it's displayed on the screen. The program then goes back to looking for another start pulse for the next character.

If your MicroBee is a BASIC-only version, getting the program into it could be a problem. Although there are only 139 bytes in the program, they can only be entered in memory by 'poke' statements, but then there's no easy way to save the program on tape. If you want to try it anyhow, first convert the hexadecimal values in the 'code' column, eight bits at a time, to decimal. Then poke them into the hex addresses in the 'ADDR' column.

Then again, you could take the easy way out. For the sum of 12 miserable dollars, sent to the author, you will receive a postpaid cassette tape which can be loaded into your MicroBee with the usual 'load' command.

This tape contains an extended 'bells and whistles' version of the RTTY program, with such goodies as baud rates from 45.45 to 300, selectable while the program is running, selectable page or tape display mode, and signal inversion at the touch of a key.

Construction

All the 'electronic' bits go on a pc board measuring about 120 mm by 60 mm. Only the tuning meter and input/output connectors are mounted separately. As a concession to the 'RF purists' I used a double-sided pc board with a groundplane on the component side of the board. As we're dealing only with audio signals here, you can ignore the groundplane if you so wish. As for sockets for the two ICs — please yourself, they have no effect on circuit operation.

Assembly of the pc board is straightforward. Probably the easiest way to tackle it is to solder the resistors in place first, followed by the capacitors. Watch the orientation of the 1 uF tantalum, C5. Install IC1 and then IC2. Note that the latter is a CMOS type and the usual static and soldering precautions should be taken. Note that there is one link on the board — near one end of IC2.

Finally, attach the wires that go to the meter and input/output terminals. If using a double-sided pc board, some components are soldered on the top *and* bottom side of the board — denoted by a • on the overlay.

You can mount the decoder in a box if you wish; any suitably-sized jiffy box will do nicely. The meter and input/output terminals can be mounted on the box's lid. Nothing's critical, so exact construction details are left up to you as individual requirements will undoubtedly vary a great deal.

Hooking it up

Figure 2 shows the general idea of how the

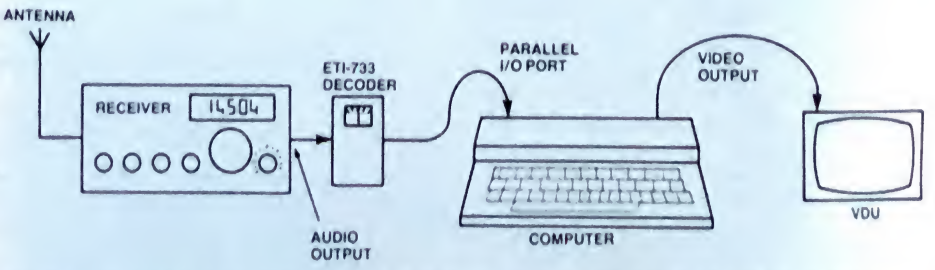


Figure 2. The hookup. Audio output from the receiver can be taken from across the speaker, a headphones output, a 'recorder' output or other suitable auxilliary audio output.

PARTS LIST — ETI-733

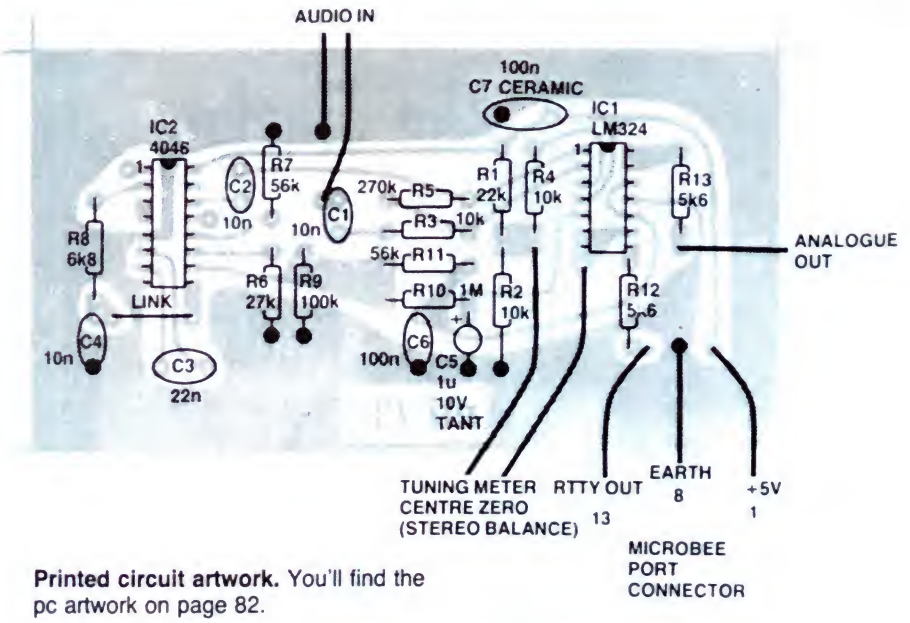
Resistors ..... all ½W, 5%
R1 ..... 22k
R2, 3, 4 ..... 10k
R5 ..... 270k
R6 ..... 27k
R7, R11 ..... 56k
R8 ..... 6k8
R9 ..... 100k
R10 ..... 1M
R12, R13 ..... 5k6

Capacitors
C1, 2, 4 ..... 10n greencaps
C3 ..... 22n greencap
C5 ..... 1u/10 V tant.
C6 ..... 100n greencap
C7 ..... 100n ceramic bypass

Semiconductors
IC1 ..... LM324
IC2 4046

Miscellaneous
ETI-733 pc board; M1 — centre-zero tuning meter; DB15 plug, wire, etc.

Price estimate $16 — $20



Printed circuit artwork. You'll find the pc artwork on page 82.

decoder fits in the system. The audio output from the receiver can be taken directly across the speaker, from any auxiliary audio output or a headphones output. Less than one volt of audio will provide adequate drive to the decoder.

To use a MicroBee computer you'll first have to arrange a parallel input/output port. If this was not supplied with your computer, you'll have to install it yourself. All that's required is the 15-pin ('DB15S') socket which you can buy from your local electronics store and wire it in yourself (very carefully). While it wires straight in, note that bit 0 of the port comes out on pin 13, not pin 11, as the pc board overlay or kit construction manual indicates. The output of the ETI-733 decoder connects to bit 0 and it gets power supply from the computer via pin 8 of the port connector.

All you need now is some means of converting the decoder output to characters on the VDU screen. That's where we need a little software.

For other Z80-based computers, we'll have to leave the details to you as individual systems vary. Have a close look at your system's technical manual.

On the air

To test your new decoder, here are some frequencies to try:

11 030 kHz AXM, coded weather info, 50 bauds
13 779 kHz Voice of America News, 75 bauds
14 700 kHz Christchurch to McMurdo, 75 bauds
16 100 kHz Chinese News Agency, 50 bauds

With the receiver set for the upper sideband mode, tune for centre reading on the decoder meter. If the signal is garbled, try it on the other sideband. An off-centre meter reading under no-signal conditions is normal. Happy spying. ■

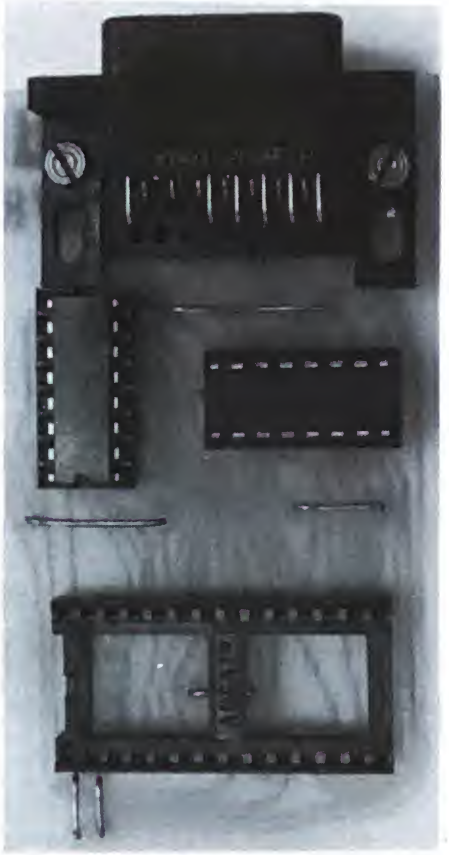
SOFTWARE

```
ADDR   CODE    LINE   LABEL  MNEM    OPERAND

               00100 ;RTTY RECEPTION PROGRAM FOR 50 BAUDS
               00110 ;(SHORT VERSION) by Tom Moffat
               00120
0400           00130          DEFP    16      ;ASSUME HEX NUMBERS
0400           00140          ORG     0400
               00150
               00160 ;5-bit serial input routine, through PIO BIT 0
               00170
0400 3ECF      00180          LD      A,0CFH  ;SET PIO FOR "CONTROL"
0402 D301      00190          OUT     (1),A
0404 3EFF      00200          LD      A,0FFH  ;SET ALL BITS "INPUT"
0406 D301      00210          OUT     (1),A
0408 DB00      00220 INPT     IN      A,(0)   ;LOOK FOR START BIT
040A CB47      00230          BIT     0,A
040C 20FA      00240          JR      NZ,INPT
040E 0E08      00250          LD      C,8     ;SET UP COUNTER BIT
0410 CD4204    00260          CALL    DELAY1  ;START BIT IS LOW, ...
0413 CD3F04    00270          CALL    DELAY2  ;DELAY TO CENTRE OF DB-1
0416 DB00      00280 LOOP     IN      A,(0)   ;BRING IN DATA BIT
0418 CB3F      00290          SRL     A       ;SHIFT DATA INTO CARRY
041A CB11      00300          RL      C       ;SHIFT CARRY INTO C
041C CD3F04    00310          CALL    DELAY2  ;ONE BIT TIME
041F 30F5      00320          JR      NC,LOOP ;RPT TILL COUNTER BIT OUT
               00330
               00340 ;Conversion to ASCII, BAUDOT character in C.
               00350
0421 214C04    00360          LD      HL,TABLE
0424 0600      00370          LD      B,0
0426 1600      00380          LD      D,0
0428 79        00390          LD      A,C
0429 FE1B      00400          CP      1BH     ;IS BAUDOT CODE 'FIGS' ?
042B 2002      00410          JR      NZ,$+4  ;SKIP NEXT IF NOT FIGS
042D 1E01      00420          LD      E,1     ;SET FLAG "FIGS"
042F FE1F      00430          CP      1FH     ;IS BAUDOT CODE 'LTRS' ?
0431 2002      00440          JR      NZ,$+4  ;SKIP NEXT IF NOT LTRS
0433 1E00      00450          LD      E,0     ;SET FLAG "LTRS"
0435 CB21      00460          SLA     C       ;LET C = C * 2
0437 19        00470          ADD     HL,DE   ;ADD FLAG TO POINTER
0438 09        00480          ADD     HL,BC   ;ADD CHR TO POINTER
043A 46        00490          LD      B,(HL)  ;ASCII CHAR INTO B
043B CD0C00    00500          CALL    000CH   ;OUTPUT CHAR TO VDU.
043E 18C9      00510          JR      INPT    ;GO BACK TO START
               00520
               00530 ;Time delay subroutine
               00540
043F CD4204    00550 DELAY2   CALL    DELAY1

0442 3E79      00560 DELAY1   LD      A,120D  ;DO 120 TIMES FOR 50 BAUD
0444 060B      00570 LOOP1    LD      B,0BH
0446 10FE      00580          DJNZ    $       ;JUMP TO YOURSELF IF NZ.
0448 3D        00590          DEC     A
0449 20F9      00600          JR      NZ,LOOP1
044B C9        00610          RET
               00620
               00630 ;ASCII lookup table arranged by BAUDOT value
               00640
044C 0505      00650 TABLE    DEFW    0505    ;(BLANK)
044E 5435      00660          DEFW    3554    ;T 5
0450 0D0D      00670          DEFW    0D0DH   ;(CR)

ADDR   CODE    LINE   LABEL  MNEM    OPERAND

0452 4F39      00680          DEFW    394FH   ;O 9
0454 2020      00690          DEFW    2020    ;(SPACE)
0456 4800      00700          DEFW    0048    ;H (STOP)
0458 4E2C      00710          DEFW    2C4EH   ;N ,
045A 4D2E      00720          DEFW    2E4DH   ;M .
045C 0A0A      00730          DEFW    0A0AH   ;(LF)
045E 4C29      00740          DEFW    294CH   ;L )
0460 5234      00750          DEFW    3452    ;R 4
0462 4724      00760          DEFW    2447    ;G $
0464 4938      00770          DEFW    3849    ;I 8
0466 5030      00780          DEFW    3050    ;P 0
0468 433A      00790          DEFW    3A43    ;C :
046A 563D      00800          DEFW    3D56    ;V =
046C 4533      00810          DEFW    3345    ;E 3
046E 5A2B      00820          DEFW    2B5AH   ;Z +
0470 4400      00830          DEFW    0044    ;D
0472 423F      00840          DEFW    3F42    ;B ?
0474 5327      00850          DEFW    2753    ;S '
0476 5936      00860          DEFW    3659    ;Y 6
0478 4625      00870          DEFW    2546    ;F %
047A 582F      00880          DEFW    2F58    ;X /
047C 412D      00890          DEFW    2D41    ;A -
047E 5732      00900          DEFW    3257    ;W 2
0480 4A87      00910          DEFW    874AH   ;J (BELL)
0482 0404      00920          DEFW    0404    ;(FIGS)
0484 5537      00930          DEFW    3755    ;U 7
0486 5131      00940          DEFW    3151    ;Q 1
0488 4B28      00950          DEFW    284BH   ;K (
048A 0B0B      00960          DEFW    0B0BH   ;(LTRS)
0000           00970          END
00000 Total errors
LOOP1   0444    TABLE   044C    LOOP    0416    DELAY2  043F
DELAY1  0442    INPT    0408
```

A ROM reader for the Microbee

So you've written your 32K 'Adventure' program and you're impressing all your friends with the graphics and the witty text — but they're not impressed at how long it takes to load the goldang program from tape! Then again, loading those short, useful utilities from tape takes a frustrating few minutes when you're hot to get on with the job. This project fixes that — transfer those tape programs to EPROM and load them with the ROM Reader.

Paul Leonardi



THIS PROJECT enables your favourite games or utilities to be loaded into your Microbee very quickly and cheaply. At the moment, if you want to change programs in the Microbee, you must either wait agonising minutes for an 8K program to load from cassette, or if you want it faster, a disk drive and associated software is needed (rather expensive for most of us). The ETI-673 MultiPROM board described elsewhere in this publication is another approach, but is limited to a number of programs and the program must be written for a certain area of memory.

The ROM Reader was designed to enable you to read a program previously written for tape storage and then transferred to EPROM. The project plugs into the 15-pin D-socket on the Microbee's rear apron and requires no external power. The program you wish to read into memory must have previously been put into a 2716, 2732 or a 2764 EPROM. If the ETI-668 Microbee EPROM blower is available, you can quite easily do this by reading the program from tape and copying to EPROM with a few small modifications to the code. This way, a library of EPROMs can be kept, similar to your library of tapes.

A machine code support program is necessary to read the information into memory quickly. A USR (XXXX) call from BASIC is all that is necessary to read in and execute the program as the first seven locations of the EPROM contain the type and load address information.

PARTS LIST — ETI-678

Resistors.....nil!
Capacitors.....nil!
Semiconductors
IC1.....74LS257
IC2.....4040B

Miscellaneous
ETI-678 pc board, 28-pin ZIF socket or 28-pin low insertion pressure socket; DA15P right-angle pc mount 15-pin plug; tinned copper wire for links

Price estimate: $23-$30

Details

The design is very simple, as you can see from the circuit diagram. However, I've pulled a few tricks. The Microbee's parallel I/O port has only eight signal lines. To read eight data lines from the EPROM, select each address location in turn and do whatever else is necessary, I have arranged the Microbee I/O lines to be a 4/in/4-out set, multiplexing the eight EPROM data lines onto the four Microbee input lines set up. Naturally, some 'driving' software is needed.

Driving software

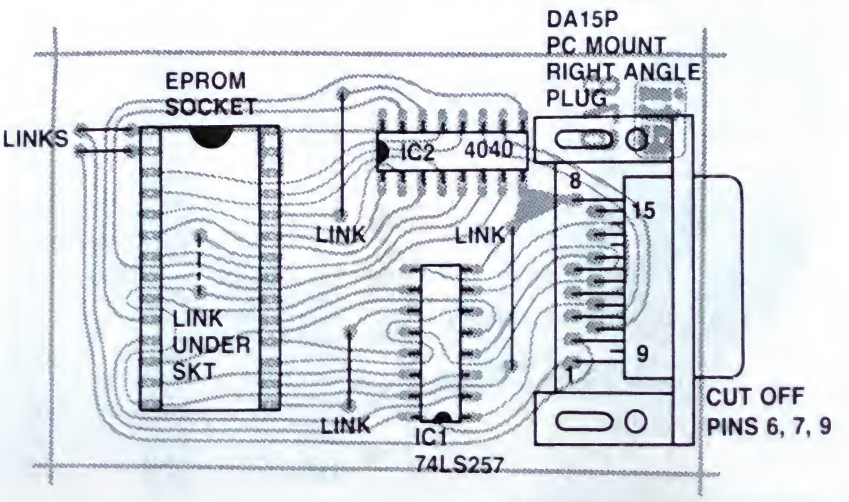
The program which enables the EPROM to be read can be ORGed to reside in RAM and thus needs to be loaded from cassette occasionally or reside in the NETWORK EPROM socket.

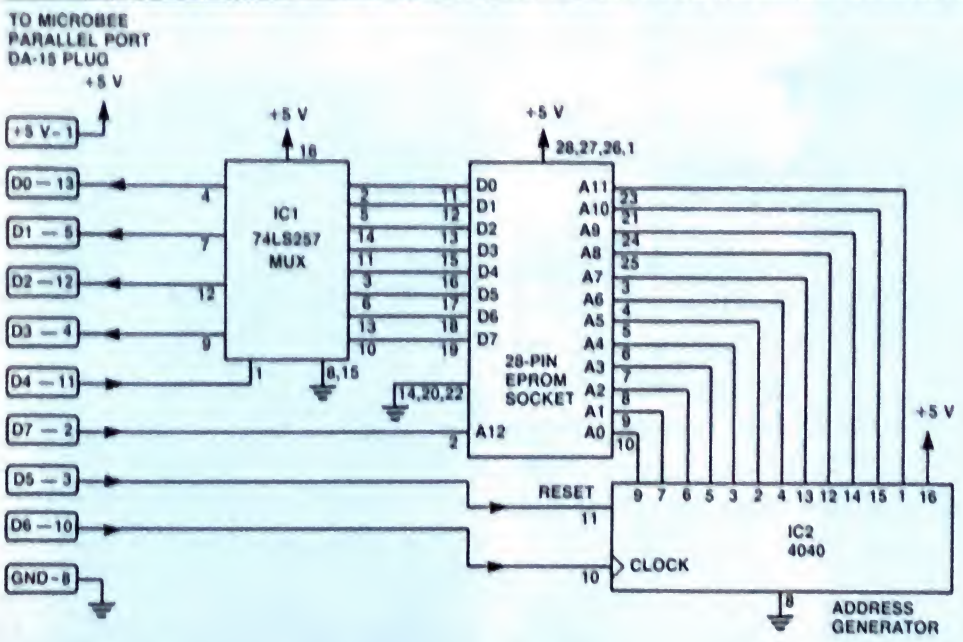
The software initially sets up the PIO for 4-in and 4-out lines. It then resets the address generator, sets A12 low and selects the least significant nibble (half a byte, four bits). The first location in the EPROM describes the EPROM type: 4 for a 2732 (4K), 8 for a 2764 (8K). If the data is zero then a 2716 is assumed. A11 is connected to Vpp and this must be high to read a 2716, so the address generator is clocked 2048 times to set A11 high. If a 2 is the first data in the EPROM then the next step is carried out, else an error message is displayed.

The 4040 is clocked to address the EPROM's next location, the data read by setting and resetting the MUX select line (pin 1) — this byte is the most significant byte of the address to where the program is to be loaded. The next location contains the least significant byte of the load address. The following two locations specify the number of bytes to be transferred and the next two contain the program start address. The actual program resides after the initial seven bytes.

Construction

Assembling the project is quite easy. First, pins 6, 7 and 9 of the DB16P need to be cut off and discarded so that the socket can be mounted to the pc board. The way tracks had to run on the board made this necessary. Carefully check the pc board to see that all holes are correctly drilled and that there are no breaks in the tracks or hairline ▶





HOW IT WORKS — ETI-678

To enable only eight I/O lines to read eight data lines from the EPROMs, reset the address generator, clock the address generator and select A12 on 2764's, the port I/O A lines are arranged to be four in and four out.

The eight data lines are multiplexed by IC1, a 74LS257, on the four input lines to the Microbee (D0-D3) and one output line (D4) is used to select the higher or lower nibble (4 bits) when reading.

Two output lines reset and clock the 12-bit ripple counter IC2, a 5050, used as an address generator. The last output is used to address A12 as the address generator only goes to A11.

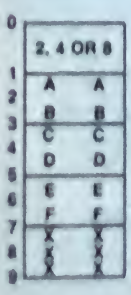
The driver software which enables the data to be read into the Microbee is described in the software section. Power is derived from the Microbee, via pin 1 of the DB15 connector.

'bridges', particularly where tracks or pads are closely-spaced. See that you can screw the DB15 plug to the board without straining the pins. Don't mount it just yet, though.

The six links need to be inserted and soldered in place first. Don't forget the one beneath the EPROM socket. While two IC sockets were used on the prototype shown in the photograph, they aren't essential. If you're using them, put them on next, otherwise, mount the EPROM socket. You have three alternatives here: the expensive way, the less expensive way and the cheap way. The expensive way is to buy a 28-pin *zero insertion force* socket. It's elegant, handy to use and saves your valuable EPROMs from harm. But they cost $15-$20. The less expensive way is to use a *low insertion force* socket. They're easy to use, the EPROMs slip in and out comfortably, reducing the risk of damaged pins, and they cost under $10. The cheap way is just to use an ordinary IC socket and take care. Use a good quality IC socket if you must go for the cheap option.

Fit the DB15 plug next, bolt it to the board before soldering the pins to avoid placing strain on the soldered joints.

Assemble the ICs to the board last of all, taking care with their orientation. The 4040 is a CMOS type, so only handle it by the ends of the package, using your thumb and forefinger, and avoid touching the pins. Better still, use a static-safe IC insertion tool. If you're soldering it in, solder pins 8 and 16 first — in that order. Use a soldering iron with a grounded tip.



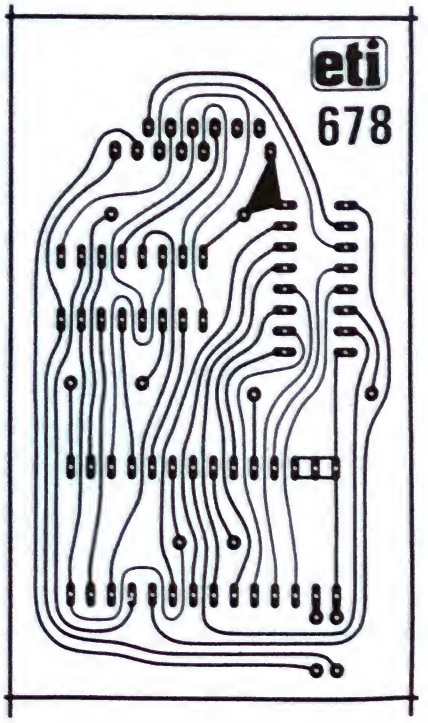
Testing

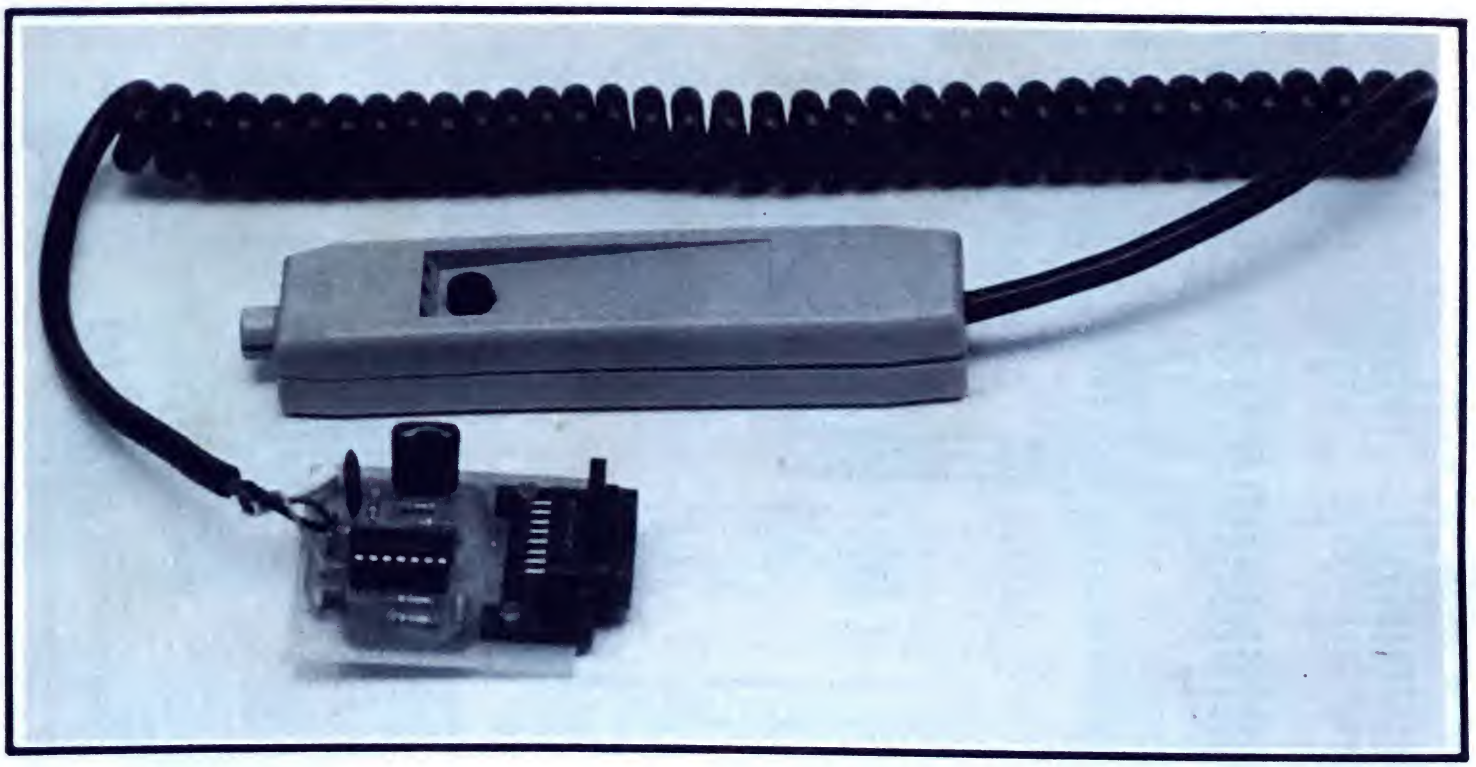
Plug the unit into the Port A socket of the Microbee. Insert a pre-programmed EPROM. If the driver software resides in the NETWORK socket, a simple MEM command from the BASIC command level will read in and execute the program, else a USR (XXXX) to where you have assembled the program will be needed.

Swift and happy ROM reading to you! ■

```
ADDR   CODE     LINE   LABEL  MNEM    OPERAND

                00100 ;****************************************
                00110 ;       EPROM READING PROG.
                00120 ;       BY P.J.LEONARDI
                00130 ;****************************************
                00140
0010            00150 BSEL    EQU     010H    ;MUX SELECT BIT
0020            00160 BRESET  EQU     020H    ;4040 RESET BIT
0040            00170 BCLOCK  EQU     040H    ;4040 CLOCK BIT
0080            00180 BAD12   EQU     080H    ;ADD. A12   BIT
                00190
                00200 ;****************************************
E000            00210         ORG     0E000H  ;E000 FOR NETWORK SPACE
                00220
                00221 ;****************************************
                00222 ;       MAIN PROGRAMM STARTS HERE
                00223 ;****************************************
                00224
E000 C303E0     00230 MEM     JP      START   ;TO PREVENT Option not fitted ERROR
E003 CD15E0     00240 START   CALL    INIT    ;INITIALIZE PIO AND 4040
E006 CD25E0     00250         CALL    TYPE    ;DETERMINE EPROM TYPE
E009 CD3CE0     00260         CALL    INFO    ;GET DESTINATION,LENGTH,START ADDR.
E00C D5         00270         PUSH    DE      ;SAVE START ADDR. ON STACK
E00D 11F90F     00280         LD      DE,4096-7       ;FIRST 7 LOCATIONS DATA
E010 CD55E0     00290         CALL    TRANS   ;TRANSFER CONTENTS TO MEMORY
E013 E1         00300         POP     HL      ;GET START ADDR. FROM STACK
E014 E9         00310         JP      (HL)    ;JUMP TO START ADDR.
                00320
                00330
                00340 ;******************************
                00350 ;       INITIALIZE
                00360 ;******************************
                00370
E015 3EFF       00380 INIT    LD      A,0FFH
E017 D301       00390         OUT     (1),A   ;SELECT PIO MODE 3
E019 3E0F       00400         LD      A,0FH
E01B D301       00410         OUT     (1),A   ;SELECT MS NIBBLE AS OUTPUT
                00420                         ;  AND  LS NIBBLE AS INPUT
E01D 3E20       00430         LD      A,BRESET
E01F D300       00440         OUT     (0),A   ;RESET THE 4040 COUNTER
E021 AF         00450         XOR     A       ;CLEAR A
E022 D300       00460         OUT     (0),A   ;CLEAR THE RESET LINE
E024 C9         00470         RET             ;INITIALIZE COMPLETE
                00480
                00490 ;****************************************
                00500 ;       EPROM TYPE DETERMINATION
                00510 ;****************************************
                00520
E025 CD6BE0     00530 TYPE    CALL    GET1    ;GET DATA FROM EPROM (1 BYTE)
E028 FE00       00540         CP      0       ;IS DATA=0
E02A CC96E0     00550         CALL    Z,A2716 ;YES, ASSUME A 2716 EPROM
E02D FE02       00560         CP      02      ;IS DATA=2
E02F C8         00570         RET     Z       ;EPROM=2716 ie. VALID
E030 FE04       00580         CP      04      ;IS DATA=4
E032 C8         00590         RET     Z       ;EPROM=2732
E033 FE00       00600         CP      00      ;IS DATA=0
E035 C8         00610         RET     Z       ;EPROM=2764
E036 21748B     00620         LD      HL,0B774H       ;ADDRESS OF "SYNTAX" MESS.
E039 CD56A2     00630         CALL    0A256H  ;PRINT ERROR MESSAGE AND GOTO BASIC
```

```
ADDR   CODE    LINE   LABEL  MNEM    OPERAND
               00640
               00650 ;****************************************
               00660 ;       PROGRAM INFORMATION
               00670 ;****************************************
               00680
E03C CD6BE0    00690 INFO    CALL    GET1    ;GET EPROM DATA AND CLOCK 4040
E03F 67        00700         LD      H,A
E040 CD6BE0    00710         CALL    GET1
E043 6F        00720         LD      L,A     ;HL=LOAD ADDRESS OF PROGRAMME
E044 CD6BE0    00730         CALL    GET1
E047 47        00740         LD      B,A
E048 CD6BE0    00750         CALL    GET1
E04B 4F        00760         LD      C,A     ;BC=LENGTH OF PROGRAMME
E04C CD6BE0    00770         CALL    GET1
E04F 57        00780         LD      D,A
E050 CD6BE0    00790         CALL    GET1
E053 5F        00800         LD      E,A     ;DE=PROGRAMME START ADRESS
E054 C9        00810         RET             ;INFORMATION COMPLETE
               00820
               00830 ;****************************************
               00840 ;       TRANSFER DATA TO MEMORY
               00850 ;****************************************
               00860
E055 CD6BE0    00870 TRANS   CALL    GET1    ;GET DATA FROM EPROM AND CLOCK 4040
E058 77        00880         LD      (HL),A  ;PUT DATA INTO LOAD ADDRESS POINTER
E059 0B        00890         DEC     BC      ;DECREMENT LENGTH TO GO COUNTER
E05A 1B        00900         DEC     DE      ;DECREMENT A12 OFF COUNTER
E05B 23        00910         INC     HL      ;INCREMENT LOAD ADDRESS POINTER
E05C 7A        00920         LD      A,D
E05D B3        00930         OR      E       ;IS DE=0000
E05E 2005      00940         JR      Z,MA12  ;YES  SET A12 HIGH
E060 78        00950 CONT    LD      A,B
E061 B1        00960         OR      C       ;IS BC=0000 IE PROGRAMM TRX COMPLETE
E062 20F1      00970         JR      NZ,TRANS        ;NO SO DO IT AGAIN SAM
E064 C9        00980         RET             ;PROGRAMME TRANSFER COMPLETE
E065 3E80      00990 MA12    LD      A,BAD12 ;A12 BIT
E067 D300      01000         OUT     (0),A   ;SET A12 HIGH ON EPROM
E069 18F5      01010         JR      CONT    ;CONTINUE WITH  TRANSFER
               01020
               01030
               01040 ;****************************************
               01050 ;       GET 1 BYTE OF DATA FROM EPROM
               01060 ;       AND CLOCK THE ADDRESS GENERATOR
               01070 ;****************************************
               01080
E06B C5        01090 GET1    PUSH    BC      ;SAVE BC ON STACK
E06C DB00      01100         IN      A,(0)   ;LOOK FOR A12 STATUS
E06E E680      01110         AND     BAD12   ;RETAIN A12,LOSE MUX SELECT AND CLOC
K
E070 D300      01120         OUT     (0),A   ;SELECT LS NIBBLE
E072 DB00      01130         IN      A,(0)   ;READ IT
E074 E60F      01140         AND     0FH     ;MASK OUTPUTS
E076 47        01150         LD      B,A     ;SAVE IN BREG
E077 DB00      01160         IN      A,(0)   ;RETAIN A12
E079 F610      01170         OR      BSEL    ;SET SELECT HIGH
E07B D300      01180         OUT     (0),A   ;SELECT MS NIBBLE
E07D DB00      01190         IN      A,(0)   ;READ IT
E07F CB27      01200         SLA     A
E081 CB27      01210         SLA     A
E083 CB27      01220         SLA     A
E085 CB27      01230         SLA     A       ;PUT MS NIBBLE IN MS NIBBLE OF A
E087 80        01240         ADD     A,B     ;ADD MS NIBBLE TO LS NIBBLE
E088 47        01250         LD      B,A     ;SAVE DATA IN BREG
E089 DB00      01260         IN      A,(0)   ;RETAIN A12
E08B F640      01270         OR      BCLOCK  ;SET CLOCK BIT HIGH
E08D D300      01280         OUT     (0),A   ;CLOCK THE ADDRESS GENERATOR
E08F E680      01290         AND     BAD12   ;REATIN A12,CLEAR OTHER OUTPUTS
E091 D300      01300         OUT     (0),A   ;RESET CLOCK BIT
E093 78        01310         LD      A,B     ;A REG = DATA NEEDED
E094 C1        01320         POP     BC      ;GET BC REG BACK
E095 C9        01330         RET             ;DATA IN A REG AND 4040 CLOCKED
               01340
               01350
               01360 ;****************************************
               01370 ;       2716 EPROM SPECIAL TREATEMENT
               01380 ;****************************************
               01390
E096 01FF07    01400 A2716   LD      BC,07FFH        ;2047 CYCLES
E099 CD6BE0    01410 A21     CALL    GET1            ;CLOCK 4040
E09C 0B        01420         DEC     BC              ;DECREMENT COUNTER
E09D 78        01430         LD      A,B
E09E B1        01440         OR      C               ;IS BC=0?
E09F 20F8      01450         JR      NZ,A21          ;NO DO IT AGAIN
E0A1 CD6BE0    01455         CALL    GET1
E0A4 C9        01460         RET                     ;A11 OF 4040 IS NOW HIGH
               01470                                 ;TO ENABLE EPROM COMPATABILI
TY
               01480                                 ;A11 ON 2716=VPP AND MUST
               01490                                 ;BE HIGH TO READ IT
               01500
               01510
0000           01520         END
00000 Total errors

A21     E099    CONT    E060    MA12    E065    A2716   E096
GET1    E06B    TRANS   E055    INFO    E03C    TYPE    E025
INIT    E015    START   E003    MEM     E000    BAD12   0080
BCLOCK  0040    BRESET  0020    BSEL    0010
```





The 'screen spotter' — a light pen for the Microbee

This simple, low cost device plugs into the Microbee's 8-bit port and gives you an 'entry' into the world of light pens and interactive software. The project has been developed from an idea submitted by a reader, Andrew Allen, of Manly Vale NSW.

Geoff Nicholls

ADDING A LIGHT PEN to your computer can open up a whole new range of possibilities to explore in software and the interaction between people and computers. This project should give Microbee owners an 'entry point' to some of those possibilities.

The video display chip in the Microbee has a 'light pen' input but this is not readily accessible for external connection. Hence, other avenues for adding a light pen had to be explored and the eight-bit port seemed like a simple way to go about it and that's where this project plugs into the 'Bee.

To get a photosensitive device to 'see' a single pixel on a VDU screen requires some pretty fancy optics, way beyond the resources of the home constructor, but detecting a single low-res graphics 'block' is no problem. For that reason, this project has been dubbed the 'screen spotter'.

Mechanically and electronically, the project presents few difficulties. The software, we'll leave to you — apart from a demonstration program, reproduced later.

Design

There are two parts to the Screen Spotter — the 'head' and the 'interface'. The head is housed in a plastic logic probe case and contains a phototransistor to detect light from the VDU screen, plus pulse-forming circuitry and a momentary-action pushbutton so you can signal a 'response' to the computer.

The interface unit is a small board mounted on a DB15 plug which fits in the Microbee's 8-bit port. This board contains circuitry which provides the appropriate signals to the computer.

The head and interface units are connected via a coiled cord of three wires plus a shield.

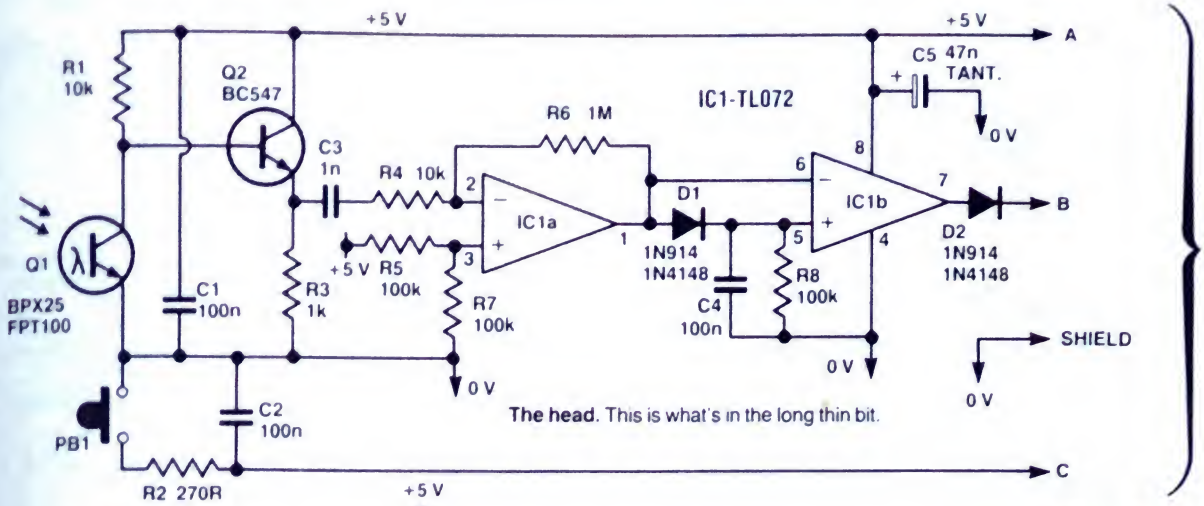
A coiled cord (rather like the one on your telephone) keeps itself out of the way when the unit is not in use. This cable does not have to be shielded, a four-wire cable will serve just as well.

Housing the head gave me a few headaches! Putting the bits in the barrel of a suitable pen is possible, but very difficult. Cigar tubes are great, but not everybody smokes cigars! If you're not fussy, a housing is unnecessary — but the ability of the phototransistor to discriminate between adjacent spots on the screen is not good without some method of restricting its 'view'.

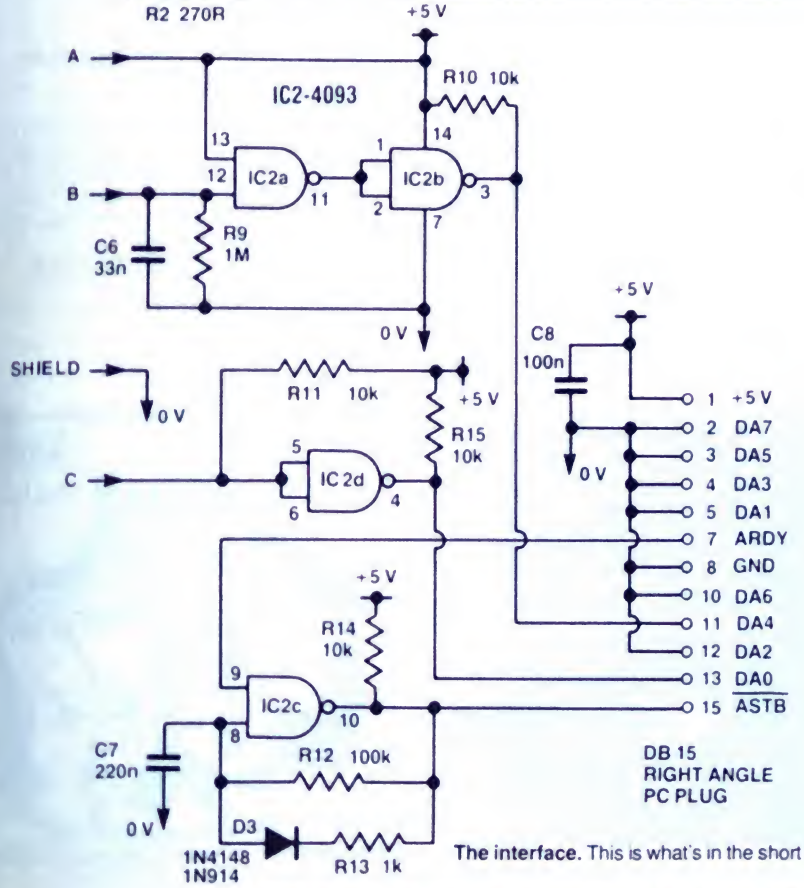
After some considerable searching and discussions with suppliers, I settled on a locally available logic probe case made by the General Specialties Corporation (USA), which was obtained from Jaycar. This comes complete with probe tip, etc, but only the case part is used. The phototransistor neatly fits in the moulding intended for the probe and this provides a tube which restricts the view of the phototransistor sufficient for the purpose here.

The pc board containing the head components was designed to fit snugly in the case. A hole has to be accurately drilled in the case top for the pushbutton key switch.

The head could be housed in some other sort of container, but that shall have to be left to your ingenuity and resourcefulness. ▶



The head. This is what's in the long thin bit.



The interface. This is what's in the short fat bit.

HOW IT WORKS — ETI 649

Phototransistor Q1 is illuminated by white areas of the video monitor and produces a corresponding variation in current through R1. Transistor Q2 buffers the phototransistor and IC1a plus associated components forms a high gain ac amplifier. IC1b and associated components square up the output from IC1a. At this point, the signal is either 0 V if the pen is on a black area of the screen, or a train of pulses if the pen is on a white area. IC2a and C6-R9-D2 stretch the pulses so that a constant '1' is obtained at IC2b if the pen is on a white area, or vice versa.

The pushbutton output is debounced by R2-C2-R11 and IC2d. If the button is pressed, the output of IC2d is 1, and vice versa.

IC2c and associated components condition the ARDY prompt from the Microbee's PIO before feeding it to the ASTB input.

The decimal value at the port input, for the four input conditions, is given in Table 1.

TABLE 1

Light pen	pushbutton	input value
on black	off	0
on black	on	1
on white	off	16
on white	on	17

THE DIFFERENCE BETWEEN THE 'SCREEN SPOTTER' AND THOSE 'BIG BUCK' LIGHT PENS

A light pen is a device that allows a computer to locate the position of a sensor placed on the face of the computer's video monitor. Light pens simplify the entry of data to the machine, allowing easy input of graphic information, selections of options from a menu or entry of moves in games, such as chess.

The performance of light pens is limited by the hardware, which boils down to "you gets what you pays for". The ETI-649 uses a very simple hardware technique which trades off speed of response for economy. There are basically two types of light pen, the complex hardware type and the complex software type. The ETI-649 is the latter.

The complex hardware light pen works by incrementing a pair of counters in sympathy with the scanning electron beam in the video monitor. One counter counts the lines on the screen while the other counts the dots (pixels) in each line. The line counter is reset at the beginning of each frame (vertical sync.) while the dot counter is reset at the beginning of each line (horizontal sync.).

The counters are read by the computer when the light pen detects the scanning beam. The sensor in this type of pen has to be very well made, with extremely fine optics, in order to resolve individual dots on the screen.

The video generator chip in the Microbee has provision for this type of light pen, but the designers of the 'Bee have cleverly used this feature to simplify the keyboard scanning circuitry. It would be possible to duplicate the light pen hardware and tap into the horizontal and vertical sync. signals inside the Microbee, but the circuit would be tricky to install and the problem of making a good sensor would remain. Perhaps some enterprising hardware buff will address the problem and develop a future ETI-XXX high resolution light pen?

The ETI-649 uses a cheap and common phototransistor to sense the light from a block on the VDU and relies on the software to scan the screen and keep track of the location. The biggest drawback is that the screen cannot be updated faster than every 20 ms because of the frame refresh of 50 Hz. Any attempt to scan faster means the video information is written to the video generator and then erased before it actually has time to output it. Although this seems a major drawback, there are techniques in programming to reduce the scanning time. No doubt readers will devise programs to utilise the project in games, etc.

Construction

The unit is quite straightforward to construct. Start with the pc boards. Whether you've bought them or built your own, check the tracks for little 'bridges' where they run close together, particularly at the IC pins, and for tiny cracks. See that all the holes are drilled and that they're the correct size. Note that the mounting hole positions for the DB15 plug will depend on the brand and type purchased.

Before assembling the head board, use it as a template to mark out the hole position for the pushbutton key switch in the probe case top. Alternatively, you could measure its position. Do this carefully and you should get the whole assembly to fit together quite easily.

Assemble the components to the head board first (ETI 649a). Solder the resistors and capacitors in place as a first step, making sure you get the electrolytic capacitor, C5, the correct way round. Mount the transistor, the two diodes and IC next, making sure you get them correctly orientated, too. If you wish, an IC socket may be used for IC1. The phototransistor is mounted at full lead length so that it may be bent over and placed in the original probe moulding. Identify its leads carefully and cut the base lead short. Last of all, solder the pushbutton key switch in place. Check it thoroughly when you've finished.

Tackle the interface board next. Install the two links first. Note that one is under IC2. Solder the resistors and capacitors in place next, followed by diode D3, making sure you orientate it correctly. An IC socket may be used for IC2. Install this or the IC next, ensuring it faces the correct way (pin 1 faces away from the DB15 connector). Now mount the DB15 plug, bolting it to the pc board *before* soldering the pins. Check it thoroughly when you've finished.

Now solder the connecting cable to the two boards. Use a pc stake or piece of tinned copper wire for the shield connection.

The probe tip moulding in the probe case should be heavily blackened with a Pentel or other marking pen before assembling the head unit. See the accompanying photograph.

Lay the head board in the probe case bottom and bend the phototransistor so that it lays in the probe tip moulding (see photograph). Arrange the cable wires so that they won't foul the assembly and then screw the probe case top in place.

Now you're ready to go.

Plug in the interface board and power up the Microbee. Use your multimeter to check that +5 V is on lead A of the cable (measured with respect to 0 V — the shield). Enter the demonstration program reproduced here and give it a try.

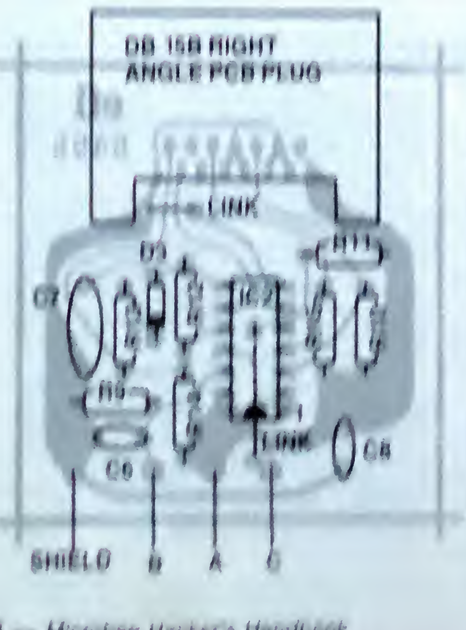
If the unit doesn't work, switch off, unplug it and look for misplaced components, any unsoldered joints or incorrectly orientated semiconductors. ■

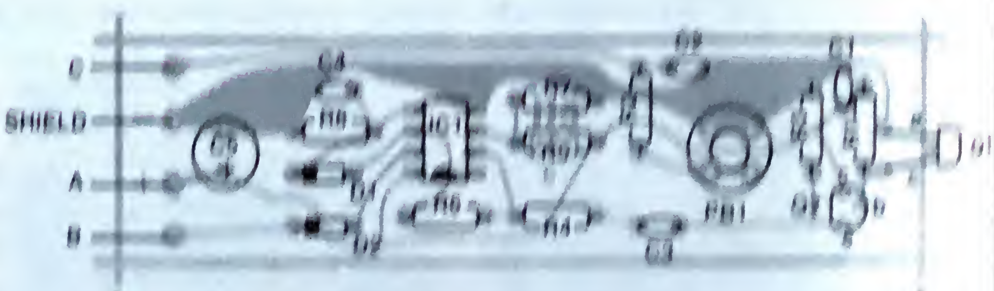
PARTSLIST — ETI 649

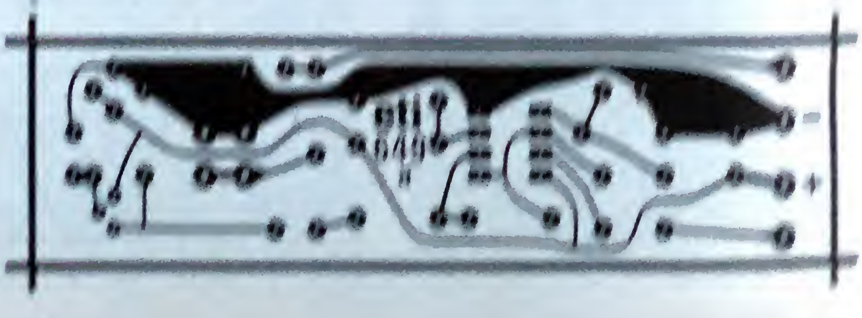
Resistors	all ¼W, 5%
R1, R4, R10, R11, R14, R15	10k
R2	270R
R3, R13	1k
R5, R7, R8, R12	100k
R6, R9	1M
Capacitors	
C1, C2	100n ceramic, bluechip
C3	1n ceramic or greencap
C4	100p greencap or ceramic
C5	47u/6V tantalum
C6	33n greencap or ceramic
C7	220n greencap or ceramic
Semiconductors	
D1, D2, D3	1N914, 1N4148
IC1	TL072
IC2	4093B
Q1	FPT100, BPX25
Q2	BC547
Miscellaneous	
SW1	keyswitch (e.g. D.S.E. S-1200 or similar)

ETI 649 pc board, case — DSE type ... or similar; DB15 plug; "curly cord" (if required) — about 1-1½ metres long, etc.

Estimated cost: $17-$35







```
00100 REM ....SCREEN SPOTTER DEMONSRATION....
00110 REM
00120 REM  This program dispays a menu of 7 music notes,
      and scans
00130 REM  them with a white block to locate the
      spotting pen.
00140 REM  The scanning block stops at the pen,and will
      play
00150 REM  the relevent note if the pushbutton is
      pressed.
00160 REM
00170 REM Clear screen and label notes.
00180 REM
00190 CLS:LORES
00200 FOR N=0 TO 6
00210  CURS 57,N*2+1
00220  PRINT CHR(N+65);
00230 NEXT N
00240 REM
00250 REM     Scanning routine loops until pen is found
      (A=16 or
00260 REM     17 if switch on),then executes the selected
      play
00270 REM     subroutine.
00280 FOR X=1 TO 13 STEP 2
00290  CURS 60,X
00300  PRINT CHR(191);CHR(191);
00310   FOR N=1 TO 6:NEXT N
00320  A=IN(0)
00330  IF A=16 THEN 320
00340  IF A<17 THEN GOTO 390
00350  GOSUB (X+1000)
00360 REM  Software debouncer
00370  A=IN(0)
00380  IF A=17 THEN 370
00390  CURS 60,X
00400  PRINT CHR(32);CHR(32);
00410 NEXT X
00420 GOTO 280
00900 REM  The play subroutines. These could be
      replaced with
00910 REM  gosub calls to execute other programs etc.
      If the
00920 REM  programs are not written as subroutines,
      then you
00930 REM  must change line 00350 to GOTO (X+1000) and
      use
00940 REM  GOTO's to the other programmes.
01001 PLAY 1
01002 RETURN
01003 PLAY 3
01004 RETURN
01005 PLAY4
01006 RETURN
01007 PLAY 6
01008 RETURN
01009 PLAY 8
01010 RETURN
01011 PLAY 9
01012 RETURN
01013 PLAY 11
01014 RETURN
02000 END
```

Project 675

RS232-to-Centronics interface

Geoff Nicholls

Most microcomputers worth owning have an 'RS232' connector, or port, through which serial communications (input/output) is conducted. It is a convention that, for listing on a printer, the BASIC LLIST or LPRINT command assumes a printer is connected to the RS232 port. Problem is, serial interface printers are more expensive than parallel 'Centronics' interface printers. Save some money, build this interface.

WHILE I designed and constructed this project to drive an Admate or TI 850 80-column dot-matrix printer from the lab. Microbee, it is 'universal' enough to suit any application requiring an interface between an RS232C port and a Centronics interface.

Printers with a parallel, or 'Centronics', interface are around $80 to $150 cheaper than with a serial, or RS232C, interface. However, the 'default' printer output on most microcomputers is via the serial interface and a serial interface printer is assumed when LLISTing or LPRINTing from BASIC. This project can be constructed for considerably less than the difference between the cost of a printer with a serial or a parallel interface.

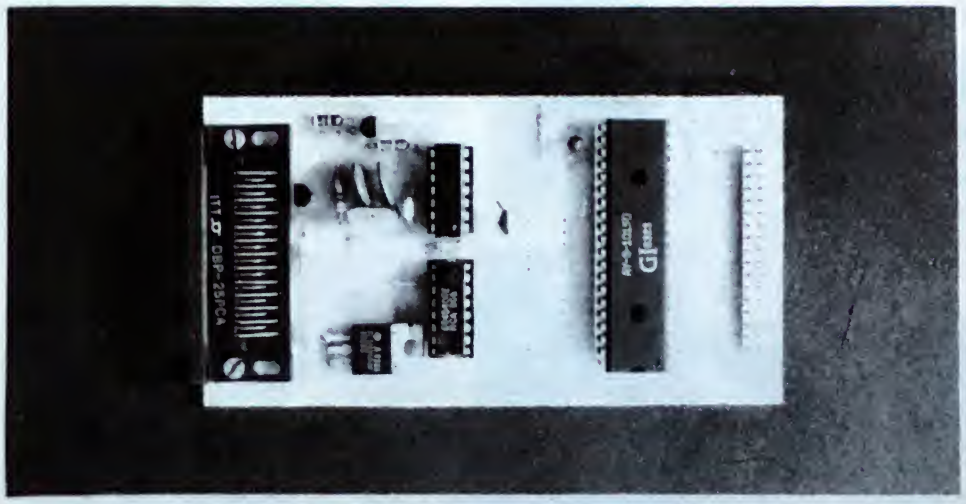
Features

The project simply plugs directly between the computer's RS232C socket and the printer's Centronics connector. It is powered from the +12 V line on the RS232C interface. It is preset to operate at a speed of 1200 baud, but provision has been made for selectable baud rates of 300, 600, 2400 and 4800 (depending on choice of one IC), in addition to that. The data format is also preset, to eight data bits one stop bit/even parity, but other formats can be selected. Tracks etched on the pc board preset the speed and data format, but provision has been made to use either links or DIL switches.

The interface is built around a single supply rail UART ('universal asynchronous, receiver-transmitter) from General Instruments, the AY-3-1015D. This chip pretty well does the whole job, even supplying the acknowledge signal (handshake) for the RS232C port. A 4.9152 MHz crystal is divided down to provide baud rate clock outputs. Either of two IC types can be used here — a 4020B or a 4040B. The 4040B provides only the lower three baud rates (300, 600 & 1200) while the 4020B provides the full complement. However, whichever type is used, it must be capable of running at 5 MHz on a 5 V supply. The minimum speed spec. for Philips and Fairchild devices equals this, but it is lower for National Semiconductor devices — though some chips may run at this speed.

Construction

Assembling the project is quite straightforward. However, there are a few points to watch. Firstly, whether you're using a ready-made pc board or have etched your own, check that all the tracks are intact and that there are no bridges, particularly where tracks run between IC pins. Also see that all the holes are drilled correctly. ▶



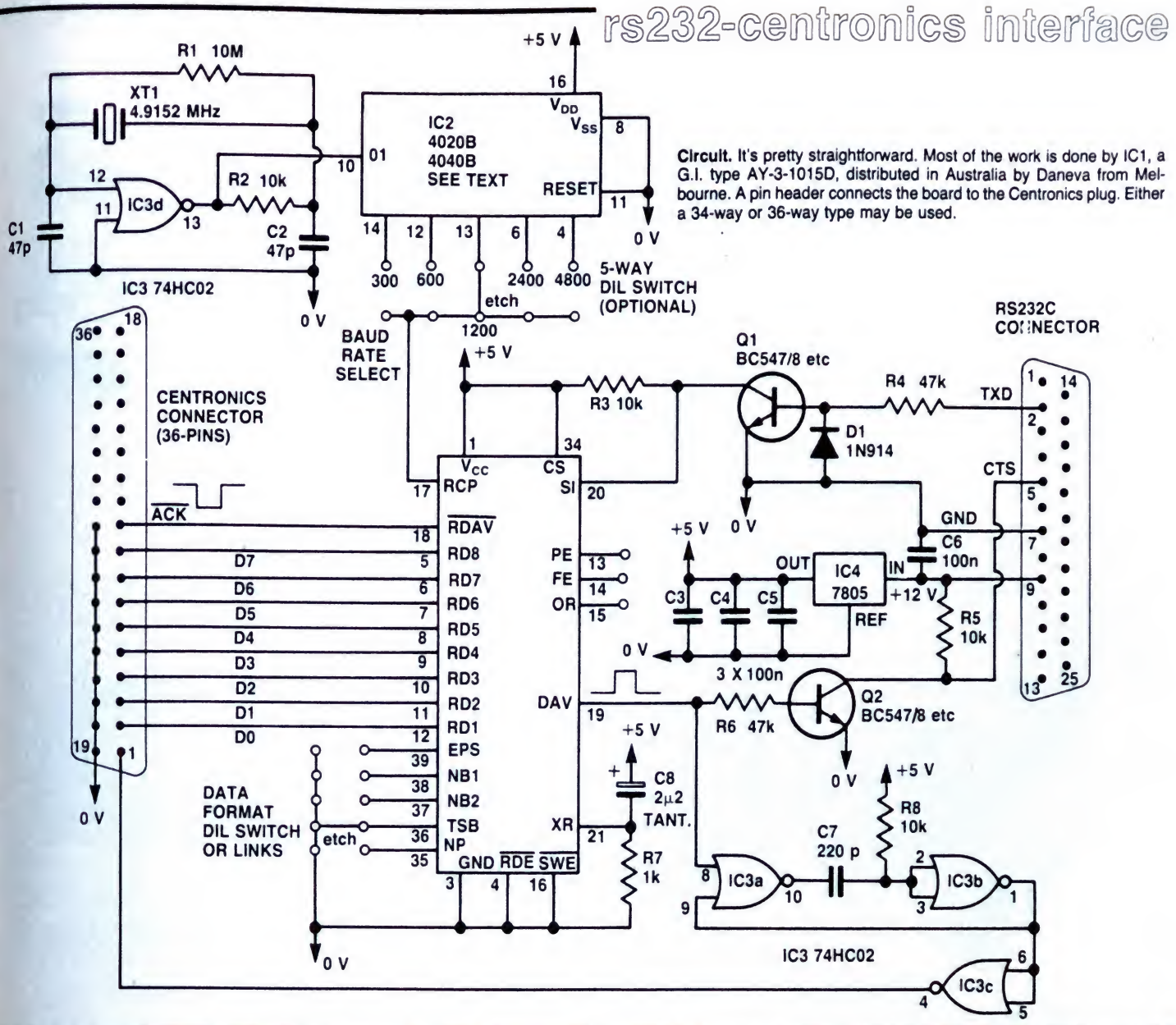
Plug in and go. There's not much to this project. The prototype here is configured to plug straight into a Microbee, but the project's suitable for any micro with an RS232C port.

HOW IT WORKS ETI-675

Most of the circuitry in the interface is inside IC1, a universal asynchronous receiver/transmitter, or UART. I have only utilised the receiver section, hence the large number of unused pins. Serial data enters the UART on pin 20 after being inverted and level shifted by Q1 to convert from RS232 voltages to TTL levels. When a complete word has been received the Data Available line (DAV) goes high, indicating that the word is ready to be read out. The DAV low-to-high transition triggers the monostable made from IC3 a, b, c. This generates the Data Strobe signal for the Centronics cable. The DAV signal also drives the Clear To Send (CTS) line on the RS232C side via inverting buffer Q2. This inhibits any further characters from the sending device (i.e: Microbee) until the printer has read the last one sent. When the printer is ready for another character it momentarily lowers the ACKnowledge line on the Centronics cable, which resets the DAV line via the UART input RDAV. This returns CTS to the high voltage and we are back where we started, ready to receive a new character.

The baud rate clock comprises IC3d, a standard crystal oscillator using a high speed CMOS gate, and IC2, a multistage divider. The divided outputs are brought to a

rs232-centronics interface



Circuit. It's pretty straightforward. Most of the work is done by IC1, a G.I. type AY-3-1015D, distributed in Australia by Daneva from Melbourne. A pin header connects the board to the Centronics plug. Either a 34-way or 36-way type may be used.

pad array, which can be used with a 5-position DIL switch to change rates if you wish. The PC board is etched to only use 1200 baud, so that Microbee users need not change anything.

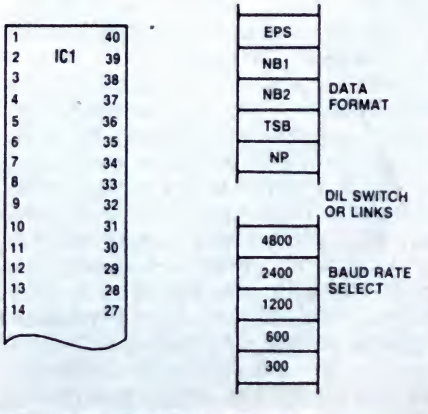
Either a 4020B or 4040B can be used for IC2, but the highest two baud rates will be different (1200 baud is unaffected). The overlay shows the rates for a 4020B i.e: 300, 600, 1200, 2400 and 4800 baud. If a 4040 is used then the lowest three rates will be the same but extra links will have to run to get 2400 and 4800 baud, owing to the different pinouts.

IC2 must be able to run with a 4.9 MHz clock at 5 V, which is slightly faster than the typical spec. for National devices, but is the minimum spec. for Fairchild or Philips (HEF4020B, HEF4040B). The symptom of a slow device is that the first stage divides by three instead of two, so look at pin 9 with a counter to make sure all is well.

The CR network of C8 and R7 reset the UART on power-up while IC4 and associated components develop the +5 V rail from the RS232C's positive supply. The pads near pins 35-39 of the UART set up the parity and bit length of the serial conversion, they are preset by board tracks to suit the Microbee.

OPTIONAL DIP SWITCHES

The PC board has been laid out to allow the option of fitting two 5-way DIL switch banks, between the UART and ICs 2 and 3. This allows the selection of different baud rates, parity and stop bits. The PC board comes with tracks etched to set eight data bits, one stop bit, no parity bit and 1200 baud. These suit most uses, but if you want to run some other combination then you will have to cut the two tracks and install wire links or DIL switches. The signal definitions are as per the diagram here and Table 1.

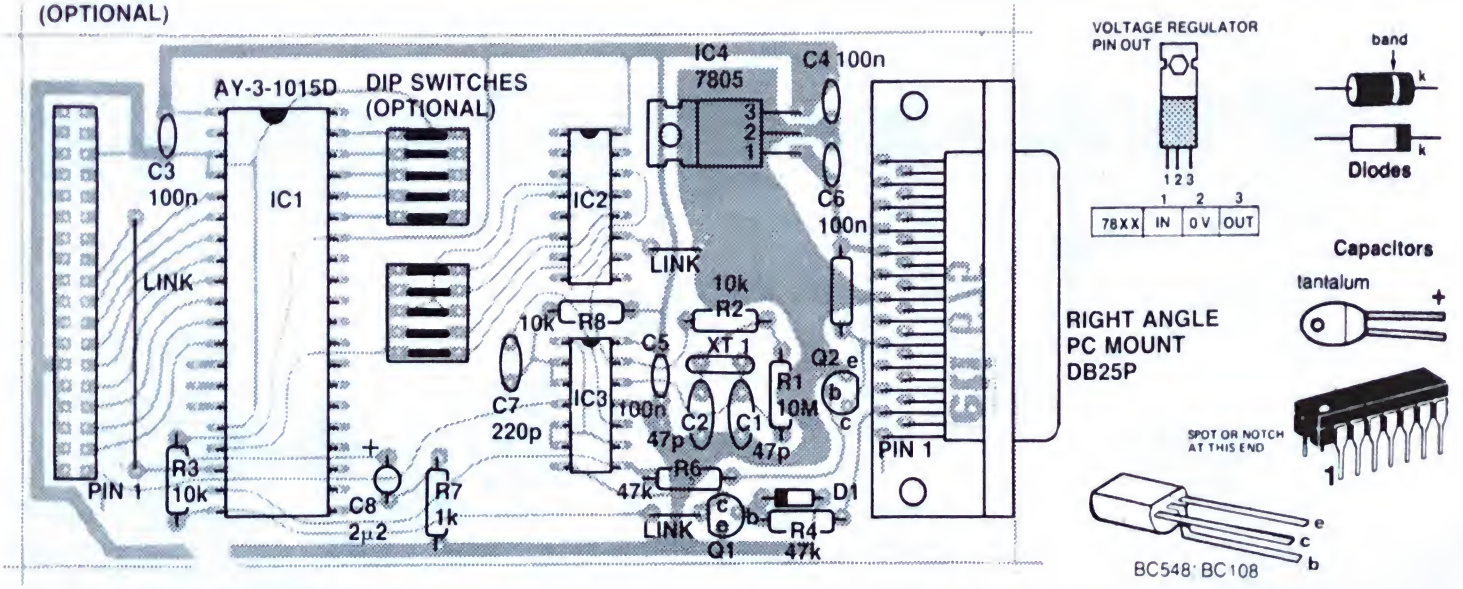


NOTE THAT YOU CAN ONLY GET 2400 AND 4800 BAUD WHEN USING THE 4020B FOR IC2

TABLE 1

SIGNAL	FUNCTION		
EPS	ON — ODD PARITY	OFF — EVEN PARITY	
NB1, NB2	NUMBER OF BITS PER CHARACTER		
	NB2	NB1	BITS
	ON	ON	5
	ON	OFF	6
	OFF	ON	7
	OFF	OFF	8
TSB	ON — 1 STOP BIT	OFF — 2 STOP BITS	
NP	ON — NO PARITY	OFF — PARITY EXPECTED	

Only one switch in the baud rate bank may be on at one time, otherwise improper operation will occur.



PARTS LIST — ETI-675

Resistors ..... all ¼W, 5%
- R1 ..... 10M (10% OK)
- R2, 3, 5, 8 ..... 10k
- R4, R6 ..... 47k
- R7 ..... 1k

Capacitors
- C1, C2 ..... 47p NPO ceramic
- C3, 4, 5, 6 ..... 100n 'bluechip' ceramic
- C7 ..... 220p ceramic
- C8 ..... 2µ2/6 V tantalum

Semiconductors
- IC1 ..... AY-3-1015D (G.I.) or exact equiv.
- IC2 ..... 4020B or 4040B (must run at 4.9 MHz on 5 V supply)*
- IC3 ..... 74HC02
- Q1, Q2 ..... BC547, BC548 etc.
- D1 ..... 1N914

Miscellaneous
- XT1 ..... 4.9152 MHz crystal, HC18/U

ETI-674 pc board; right angle pc mount DB25 plug; 34-way pin header (0.1 x 0.1") — optional; 34-way female IDC plug (optional); IDC Centronics plug (if required); 34-way ribbon cable; two 5-way DIL switches (optional); etc.

Price estimate: $28-$58

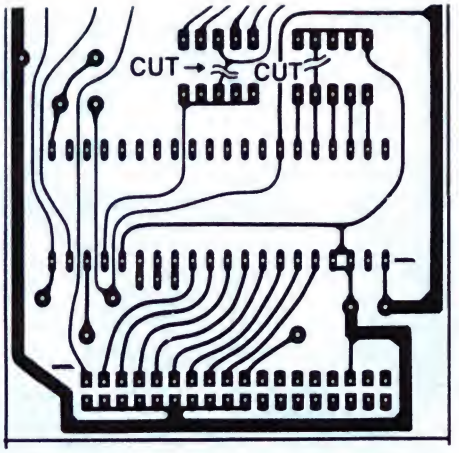
*Philips HEF4020B/HEF4040B and Fairchild 4020B/4040B known to work.

Solder all the resistors and capacitors in place first. Make sure you get the tantalum capacitor (C8) the right way round. Next solder D1 in place, making sure you get it the right way round, too. Follow with the two transistors, Q1 and Q2.

Solder the three links in place next, followed by the three IC sockets and the crystal. The three-terminal regulator (IC4) can be soldered in now. Note that a heatsink is not required for it. If you are using DIL switches for baud rate and data selection, these should be soldered in place next.

Last of all, solder the 34-pin header strip and DB25P right angle connector in place. The latter should be bolted to the board firmly before soldering so that no stress is placed on the solder joints. Insert the ICs now. The UART is an NMOS device, while the other two are CMOS. Take the usual precautions against static damage.

Make up interconnecting cables to suit your individual requirements (see next section). If you're using a Centronics plug, I strongly recommend you use an insulation displacement type as the solder pin type is much harder to assemble, with a chance of errors and poor connections.



A cut above. To obtain other baud rates and data formats, the two tracks marked above must be cut. Links or DIL switches can be used to select the required configuration.

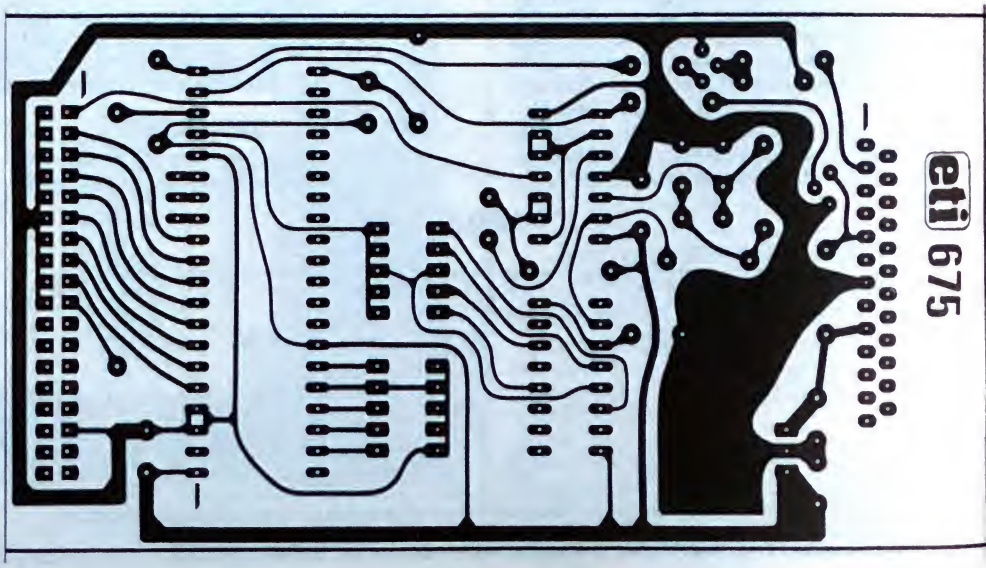
Mounting

Actual mounting details are left up to you, as requirements will vary widely. The board can be mounted inside your printer, in which case the DP25P plug and 34-pin header may be dispensed with and the board wired-in directly. Alternatively, the board may be mounted inside a zippy box, or other suitable case, and cables wired to it with suitable connectors on the end. Or, if you have a Microbee, you can do as I did and plug it straight in to the DB25 socket on the rear of the cabinet and let it hang out the back. A couple of 'feet' might be useful, though. These could consist of two standoff pillars with rubber grommets attached, bolted to the end of the board either side of the 34-pin header. The overall price of the project depends on the connectors and case used — or not used.

Using it

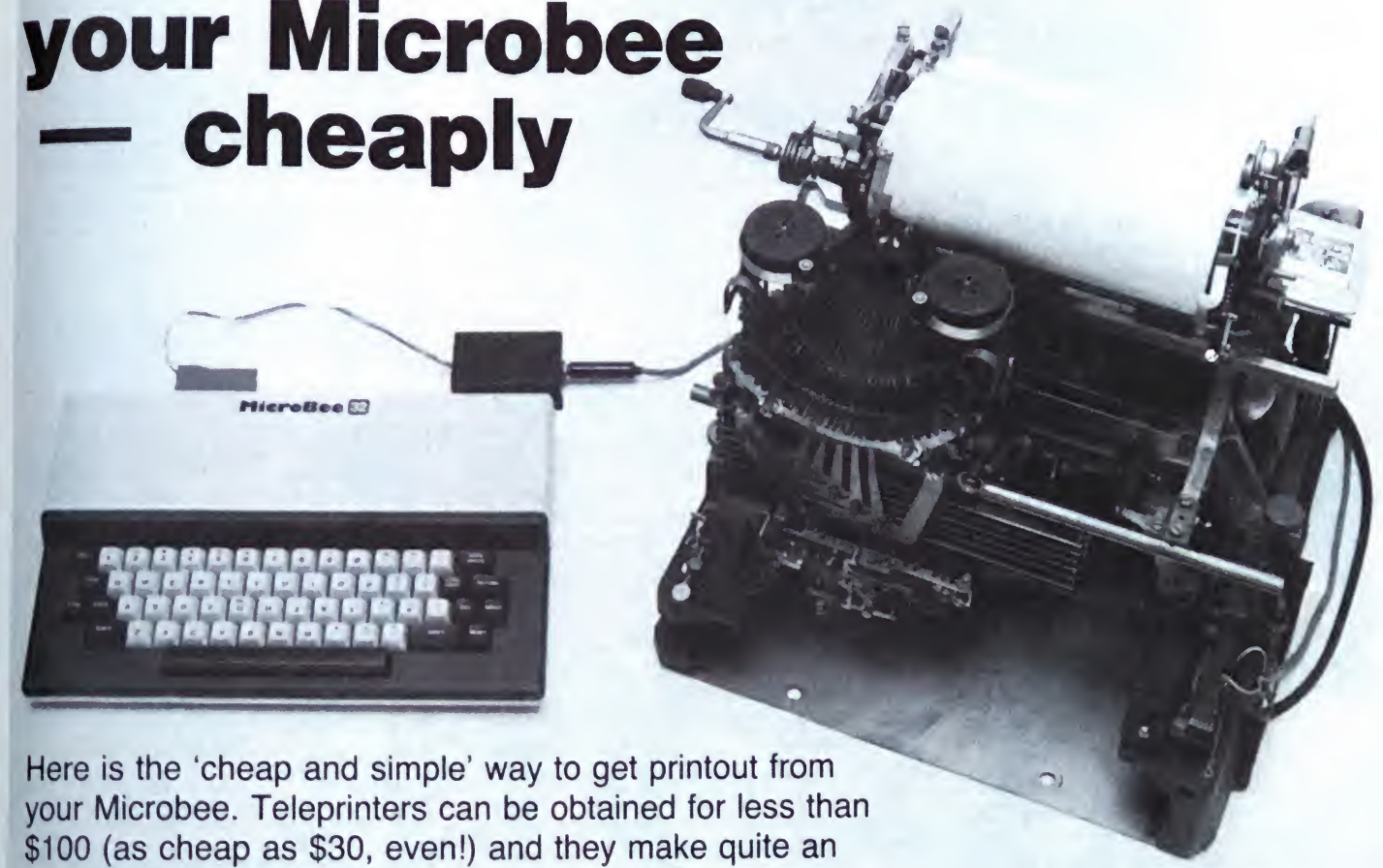
With a Microbee, just plug it in and go! With other micros, setting the baud rate and data format is simply a matter of cutting the two tracks (as indicated in the diagram here) and either installing links or DIP switches and linking the appropriate outputs across as per Table 1.

Happy LLISTing! ■



Board artwork. Full-size reproduction of the pc board. ▶

Getting hard copy from your Microbee — cheaply



Here is the 'cheap and simple' way to get printout from your Microbee. Teleprinters can be obtained for less than $100 (as cheap as $30, even!) and they make quite an acceptable substitute if you can't afford a 'real' printer. This project interfaces your Microbee to a teleprinter.

Tom Moffat
39 Pillinger Drive, Fern Tree, Tas. 7101

YOU MAY RECALL that some of the early Microbee programs in ETI were originally printed out on an old 50 baud teleprinter. This worked well for assembly language programs, but there were a few problems when BASIC was being printed. One program even had to be published a second time with a proper ASCII printout because people couldn't make much sense of the first version. The trouble is the Baudot character set . . . some of those special characters such as "*" and "#" just don't exist. But even with all the troubles, I still keep getting letters and phone calls asking "how'd you do it?"

I did it, originally, with a second computer, a small 6800-based machine. The machine code program took in ASCII material at 1200 bauds and translated it to Baudot code which was sent to the printer at 50 bauds. The data was transferred a line at a time, and then the Microbee was forced to wait while the other computer drove the teleprinter. The system looked like any 1200 baud printer to the 'Bee, and the signals came out of the second computer as audio tones which were decoded with an ETI-730 radioteletype converter. Nothing had to be modified, everything just plugged together, but it was messy. Now, due to all that popular demand, here's a teletype machine driver system that's internal to the Microbee.

Using the 50-baud printing system described here is simplicity itself. All you do is play a tape into the 'Bee and from then on anything that would have come out the RS232 port as 1200 baud ASCII comes out as 50 baud Baudot instead. A small interface circuit then converts the RS232 data levels to high voltage drive for the teleprinter. Because of the Microbee's battery backup system, the 50 baud routine remains in force until you cold-start the BASIC. That re-initializes the 1200 baud routine, which stays in force until you re-load the tape.

How I did it

The signals at the 'Bee's RS232 port are 0 V for mark and +10 V or thereabouts for space. Not quite RS232 standard, but it seems to drive printers all right. The signals are upside-down for teletype, and of the wrong voltage. The circuit shown in Figure 1 will take care of this. I built a prototype to fit inside a standard RS232 (25-pin) plug shell of the Cannon variety (the big black one). The circuit board for this was 709.68 millimetres square (or just over one square inch, to be less metric about it). As this would provide some difficulties for the average constructor, the final prototype was built on a somewhat larger board and housed in a small jiffy box.

You may notice that the design is very ▶

similar to the rear end of the ETI-730 teletype converter. In fact, if you have one of these converters you can use it instead by removing the link between R26 and R27 and feeding signals from the 'Bee's RS232 port into the driver transistor (Q2) for the loop switch at the 'free' end of R27.

Once again, the loop circuit is a direct pinch from the ETI-730 project, and if you have a loop power supply already, use it. Otherwise, you can build one up from a disused transformer, some diodes, and a filter capacitor. Perhaps you can fit it in the teleprinter's case.

Software

The first part of the program, up to line 200, fools the Microbee into thinking it's got a 1200 baud printer connected. This is the only part of the program that actually runs; it substitutes a jump address to "TTYPRT" for the jump address to the 1200 baud routine stored in ROM. The 1200 baud routine is the one the Microbee always defaults to. It's what you get when you use LPRINT or specify OUT#5. It's also the routine the editor/assembler calls for any printing operations.

The rest of the TTYPRT program, from line 2130 onwards, is just one big subroutine. The character to be printed is the Z80's accumulator when the print routine is called. The character is first inspected to see if it's a carriage return, a line feed, or a form feed. Carriage return and line feed are translated directly, as is, but form feed causes 10 line feeds to be generated. The editor/assembler sends form feeds to break up its output into pages.

Any control codes other than the ones mentioned above are thrown away. Lower case characters are converted to upper case and the character, now with a value between 0 and 3F (hex), specifies a certain entry within the hook-up table starting at line 800. Each byte in the table contains the five bits of a Baudot character in its five left-most bits. The right-most bit is an indicator

TELEPRINTERS

Frederick George Creed (1871-1957), a native of Nova Scotia, is credited with being the pioneer of the teleprinter which he adapted from a Barlock typewriter, producing his first prototype in a shed he rented for 5s in Glasgow in 1897. He retained this as a mascot for the rest of his life. Creed's first instruments actually put Morse Code on a punched paper tape but this new technology was not accepted as it was much quicker than existing Morse-encoding machines and threatened to replace the many trained operators of the day.

One Charles Krumm produced a 'teletype' machine in 1907, but it was the UK Morkrum Company who successfully developed one independently and introduced it in the early 1920s. The German Siemens-Halske company also developed a machine at this time. These machines employed a then-new encoding system developed by Jean Maurice Baudot and Donald Murray (a New Zealand farmer). The 'Baudot' code, as it is now known, is a five-unit code and requires synchronous transmission and reception (See ETI, April, '83, RTTY-Computer Decoder. Figure 1, page 80).

Creed was a strongly religious man and resigned the chairmanship of his own company in 1930 at the age of 59 because his employees insisted on playing sport on the Sabbath. He continued his interest in inventions, however, a more 'colourful' one being his permanent hair dye, only ever applied once, on his own beard, which turned an indelible rainbow pattern!

Secondhand ex-government and commercial service teleprinters of various makes can be had for quite cheap prices in most Australian states. The most common ones available are the Teletype Corp. Model 15 and Siemens Model 100, although Creed Model 7s can still be found. Scanning through Mini-Mart in recent issues of ETI shows you can pick up model 15s for $50 or less and Siemens Model 100s for between $50 and $100. Both these machines can come with or without a keyboard, the latter being the cheaper. For computer printout purposes, a keyboard is not necessary.

They can also be found in 'surplus' electronics shops. Shop around, advertise in Mini-Mart *(it's free!)*, you should not have too much difficulty finding one.

```
ADDR   CODE     LINE   LABEL   MNEM    OPERAND

                00100 ;MICROBEE TO 50 BAUD TELETYPE PRINT ROUTINE...
                00110 ;REPLACES 1200 BAUD SERIAL ROUTINE (OUTLES).
                00120 ;         - TOM MOFFAT, 31/7/83
                00130
0400            00140           DEFR    16
0140            00150           ORG     0140H
0140 215701     00160           LD      HL,TTYPRT
0143 22BC00     00170           LD      (0BCH),HL,SUBSTITUTE JUMP ADDRESS
0146 2AA200     00180           LD      HL,(0A2)
0149 E9         00190           JP      (HL)
                00200
                00210 ;START OF ASCII TO BAUDOT CONVERSION, ASCII IN A.
                00220
0157 0601       00230 TTYPRT    LD      B,1     ;PRINT ONCE
0159 0E10       00240           LD      C,10    ;TTY (CR) CHARACTER
015B FE0D       00250           CP      0DH     ;ASCII RETURN
015D 280C       00260           JR      Z,REPT
015F 0E40       00270           LD      C,40    ;TTY (LF) CHARACTER
0161 FE0A       00280           CP      0AH     ;ASCII LINE FEED
0163 282F       00290           JR      Z,PRINT
0165 060A       00300           LD      B,0AH   ;TEN LINE FEEDS
0167 FE0C       00310           CP      0CH     ;ASCII FORM FEED
0169 2005       00320           JR      NZ,KILL
016B CD9401     00330 REPT      CALL    PRINT
016E 10FB       00340           DJNZ    REPT
0170 D620       00350 KILL      SUB     20      ;KILL CONTROL CODES
0172 D8         00360           RET     C
0173 FE40       00370           CP      40      ;LOWER CASE?
0175 30F9       00380           JR      NC,KILL ;THEN SUBTRACT 20 AGAIN
0177 21BF01     00390           LD      HL,TABLE
017A 85         00400           ADD     A,L
017B 6F         00410           LD      L,A
017C 3AFF01     00420           LD      A,(FLAG),FIGS/LTRS FLAG
017F 4F         00430           LD      C,A
0180 7E         00440           LD      A,(HL)  ;TTY CHARACTER
0181 E601       00450           AND     1       ;ISOLATE F/L FLAG BIT
0183 B9         00460           CP      C       ;COMPARE PREVIOUS FLAG
0184 2808       00470           JR      Z,PRINT-1
0186 32FF01     00480           LD      (FLAG),A
0189 B7         00490           OR      A       ;FIGS OR LTRS NEEDED?
018A 0EDA       00500           LD      C,0DAH  ;TTY (FIGS)
018C 2002       00510           JR      NZ,$+4
018E 0EFA       00520           LD      C,0FAH  ;TTY (LTRS)
0190 CD9401     00530           CALL    PRINT
0193 4E         00540           LD      C,(HL)
0194 B7         00550 PRINT     OR      A       ;CLEAR CARRY
0195 CDAC01     00560           CALL    PULSE   ;START BIT
0198 C5         00570           PUSH    BC
0199 0605       00580           LD      B,5
019B CB11       00590 PRT1      RL      C
019D CDAC01     00600           CALL    PULSE   ;5 DATA BITS...
01A0 10F9       00610           DJNZ    PRT1
01A2 C1         00620           POP     BC
01A3 37         00630           SCF             ;SET CARRY
01A4 CDAC01     00640           CALL    PULSE   ;ONE STOP BIT..
01A7 11F002     00650           LD      DE,2F0

01AA 1800       00660           JR      DELAY   ;AND A HALF.
                00670
01AC DB02       00680 PULSE     IN      A,(2)   ;READ PIO
01AE CBAF       00690           RES     5,A     ;IF LOW BIT
01B0 3002       00700           JR      NC,$+4
01B2 CBEF       00710           SET     5,A     ;IF HIGH BIT
01B4 D302       00720           OUT     (2),A   ;SEND BACK TO PIO
01B6 11E005     00730           LD      DE,5E0  ;20 MS TIME DELAY
01B9 1B         00740 DELAY     DEC     DE
01BA 7A         00750           LD      A,D
01BB B3         00760           OR      E
01BC 20FB       00770           JR      NZ,DELAY
01BE C9         00780           RET
                00790
                00800 ;BAUDOT CHARACTER TABLE, ARRANGED BY ASCII VALUE
                00810
01BF 2039       00820 TABLE     DEFW    3920    ;(SPC) .
01C1 A129       00830           DEFW    29A1    ;' E
01C3 5981       00840           DEFW    0B159   ;$ F
01C5 D1A1       00850           DEFW    0A1D1   ;* '
01C7 F149       00860           DEFW    49F1    ;( )
01C9 B8A9       00870           DEFW    A9B8    ;+ ,
01CB 31C1       00880           DEFW    0C131   ;- .
01CD 39B9       00890           DEFW    0B939   ;/ /
01CF 69E9       00900           DEFW    0E969   ;0 1
01D1 C9A1       00910           DEFW    A1C9    ;2 3
01D3 5109       00920           DEFW    0951    ;4 5
01D5 A9E1       00930           DEFW    0E1A9   ;6 7
01D7 6119       00940           DEFW    1961    ;8 9
01D9 7131       00950           DEFW    3171    ;: ;
01DB F179       00960           DEFW    79F1    ;< =
01DD 4999       00970           DEFW    9949    ;> ?
01DF D1C0       00980           DEFW    0C0D1   ;@ A
01E1 9870       00990           DEFW    7098    ;B C
01E3 9080       01000           DEFW    8090    ;D E
01E5 B058       01010           DEFW    58B0    ;F G
01E7 2860       01020           DEFW    6028    ;H I
01E9 D0F0       01030           DEFW    0F0D0   ;J K
01EB 4838       01040           DEFW    3848    ;L M
01ED 3018       01050           DEFW    1830    ;N O
01EF 68E8       01060           DEFW    0E868   ;P Q
01F1 50A0       01070           DEFW    0A050   ;R S
01F3 0AE0       01080           DEFW    0E00A   ;T U
01F5 78C8       01090           DEFW    0C878   ;V W
01F7 88A8       01100           DEFW    0A888   ;X Y
01F9 88F1       01110           DEFW    0F188   ;Z [
01FB B949       01120           DEFW    49B9    ;/ ]
01FD D1D1       01130           DEFW    0D1D1   ;(UP-ARROW) (LEFT-ARROW)
                01140
01FF 00         01150 FLAG      DEFB    0
                01160
0000            01170           END
00000 TOTAL ERRORS

DELAY   01B9    PRT1    019B    PULSE   01AC    FLAG    01FF
TABLE   01BF    KILL    0170    PRINT   0194    REPT    016B
TTYPRT  0157
```

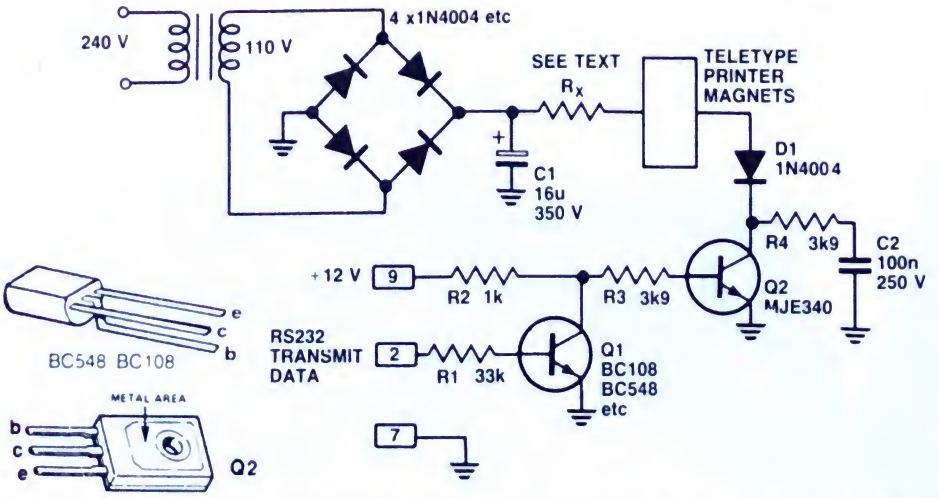
HOW IT WORKS — ETI-672

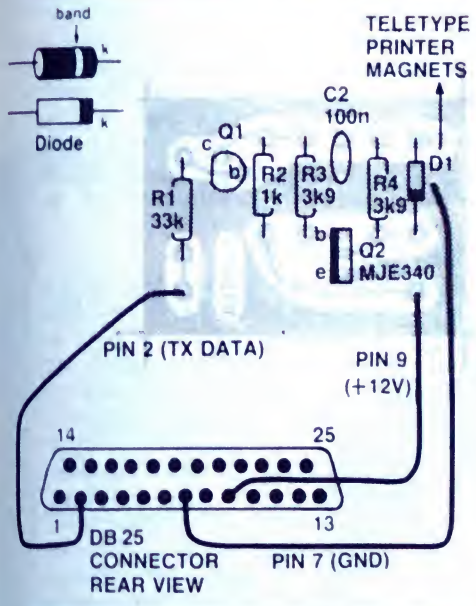
It's really quite simple. The modified-to-Baudot data signals on pin 2 of the Microbee's RS232 port drive the base of Q1 via R1, turning it on and off. The collector load for Q1 is provided by R2, the collector supply being derived from pin 9 of the RS232 port (±12 V). The signal at the collector of Q1 is thus an inverted version of the signal at pin 2 of the RS232 port, swinging from about +12 V for 'mark' to less than 1 V for 'space'.

The collector signal of Q1 drives the base of Q2 via R3. This transistor is connected in series with the teletype printer magnets via D1, thus operating the printer in accordance with the code sent.

Diode D1 and the RC network of R4-C2 smooth out inductive 'kicks' from the printer magnets.

The loop supply comprises a 110 V transformer, a diode bridge rectifier and a smoothing capacitor, C1. Resistor R_x is set to limit the current through the printer magnets to about 60 mA.





PARTS LIST — ETI-672

Resistors
- R1 ..... 33k
- R2 ..... 1k
- R3, R4 ..... 3k9

(all ¼W, 5%)

Capacitors
- C1 ..... 16µ 350 V electro.
- C2 ..... 100n 250 V greencap or ceramic.

Semiconductors
- D1 ..... 1N4004
- Q1 ..... BC108, BC548 etc
- Q2 ..... MJE340

Miscellaneous

ETI-672 pc board; UB5 jiffy box (28x54x83 mm); DB25 connector (for Microbee RS232 port); jack plug for teleprinter; teleprinter (see panel in text); loop supply components — transformer with 110 Vac or 125 Vac secondary rated at 100 mA, 4 x 1N4004, 25 W resistor (see text); hookup wire, etc.

Estimated cost: $14-$16
(less loop supply)

that tells whether the character should be figures- or letters-shifted. The bit is compared with a figures/letters flag that tells whether the previous character was in figures or letters case. If different, figures or letters is sent to the printer, and the new flag condition is stored. If the bit and the flag were the same, this part is skipped over.

The Baudot character to be printed is now copied into register C, where it's rotated out the left hand end bit by bit, with 20 millisecond time delays inserted in between. A start bit (always low) is sent before the first rotation and a stop bit (always high) is sent for 30 ms after the last data bit. With the character complete the TTYPRT subroutine returns to the calling program.

The times specified are for a 50 baud printer, but other speeds can be used as well, by changing the data in some memory locations. The Microbee IC model, with its higher speed clock frequency, is also catered for. See Table 1.

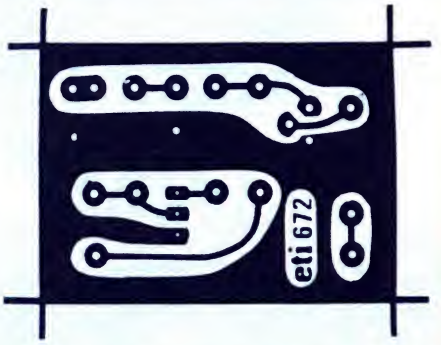
Table 1.

BAUDS	CLOCK	01A8	01A9	01B7	01B8
75	2 MHZ	F5	01	EA	03
50	2 MHZ	F0	02	E0	05
45	2 MHZ	3B	03	76	06
75	3.375	4E	03	9C	06
50	3.375	F5	04	EA	09
45	3.375	74	05	E8	0A

Construction

Construction is quite straightforward. First check the pc board has no problems — no broken tracks or tiny copper 'bridges', all holes drilled correctly, etc. If all's well, insert all the resistors and the one capacitor according to the overlay shown in Figure 2. Next insert the diode, making sure you get it the right way round (otherwise your teleprinter won't operate at all), followed by the two transistors, also making sure you get those the right way round.

The DB25 plug can then be wired up with a short cable to run to the pc board, plus a two-wire cable to a jack plug for the teleprinter. Don't wire them to the pc board yet. Drill two holes, in either end of a small jiffy box (a 'UB2' does nicely, it measures just 28 x 54 x 83 mm) and pass the two cables through, tying a knot to prevent them pulling out and leaving enough wire to



solder to the pc board. Solder the wires to the pc board. If you like, the board can be left loose in the jiffy box or, taped down with a piece of double-sided sticky pad under the Q1 end of the board. Screw the lid on and you're ready to go.

Before you start printing, however, you may need to adjust the value of R_x for correct operation of the printer magnets. Connect a milliammeter of 100 mA full scale, or a multimeter set to read 100 mA, between the anode of D1 and ground then power up the printer loop supply. Adjust R_x to get a reading of around 60 mA. An old wirewound 'slider' resistor of about 2000 or 2500 ohms, rated at 20 W or 25 W, is just right for this job, otherwise you'll have to do it by substitution. *Don't adjust the resistor with the power on and remember to discharge CI after you switch the power off and before you adjust R_x.*

Be happy, print cheaply

This whole system has been under test for quite a few weeks and it hasn't missed a beat (so far). The memory area for the program was chosen to keep it out of harm's way from other programs that may be in the computer. After all, it's meant to be an accessory, not the star attraction.

If you can lay your hands on an old teleprinter, this system is a good, clean solution to the high cost of 'proper' printers. The program listing with this article was printed by an old Siemens Model 100 teleprinter. Not bad, huh? ■

The versatile Machines can now run the CHIP-8 language, making a huge range of high-speed arcade games available to the enthusiast for no more cost than the time it takes to key them in. And the graphics potential it gives the machine is staggering.

Lindsay R. Ford

Microbee CHIP-8:

Towards the Ultimate GRAPHICS/GAMES Machine

programming. The original versions employed a 64x32 screen, but this was subsequently extended in the ETI-660 to 64 x 48 and later (when various modifications were suggested to the '660 in the Feb. '84 issue of ETI) to 64x64.

Unfortunately, this resulted in software incompatibility between machines, a factor that is of no consequence in the Microbee version as a simple 'reformat screen' instruction allows it to run software written for any of the three screen layouts.

The actual display pattern is created on the screen by the simple expedient of taking from one to 16 bytes of memory (pointed at by the 'Index' register) and transferring them to the screen at co-ordinates specified by two variables (see Figure 1). The resultant binary pattern can be used to create a multitude of shapes and has the advantage of being extremely fast. The display instruction also aids animation by allowing selective erasure of existing displays and can detect when two displayed objects overlap (for instance, allowing the program to detect when the 'laser' hits the 'alien'.)

Other features supported by the versions of the language now in use in Australia (which we shall refer to as the 'core language') include a simple tone generator, a random number generator and commands to carry out arithmetic and logic operations. Figure 2 lists the 'core' language instructions.

Extended CHIP-8

The new version of CHIP-8 fully implements the core language, but its 'user selectable' screen format means that the bulk of software written for earlier machines can be run with little or no modification. An added advantage for the Microbee owner is that the interpreter extends several of the language subsets to take advantage of some of the 'Bee's more sophisticated features (and even a few it didn't originally have!! — see Figure 3). This will allow programmers to avoid the complexities of assembly language, yet still have virtually unlimited scope to write fast action games. The new instructions cover;

Hi-res Graphics: One of the most consistent requests of CHIP-8 users over the years has been for some method of obtaining high resolution graphics. Several people have tried to implement this by introducing changed screen formats (eg: 128x64), but this always results in reduced speed and the loss of the useful high speed 'chunky graphics' that were the main feature of the language in its original form. The new Microbee interpreter side-steps these difficulties by retaining standard graphics in the three main screen formats, but supplements them with 112 cursors (the standard Microbee PCG graphics cursors) that can be loaded with data and displayed using two simple commands. This allows graphics with a fantastic 512x256 resolution to be mixed with any of the other standard and non-standard graphic styles available with this interpreter. ▶

CODE	FUNCTION
0000	NOP (No Operation — ETI-660 uses "00FF")
00E0	Clear the screen
00EE	Return (from subroutine)
0MMM	Do machine code subroutine at $MMM
1MMM	Goto $MMM
2MMM	Gosub $MMM
3XKK	Skip next instruction if VX=KK
4XKK	Skip next instruction if VX=KK
5XY0	Skip next instruction if VX=VY
6XKK	Set VX=KK
7XKK	Set VX=VX+KK
8XY0	Set VX=VY
8XY1	Set VX=VX OR VY (logical OR)
8XY2	Set VX=VX AND VY (logical AND)
8XY4	Set VX=VX+VY (VF=1 if carry)
8XY5	Set VX=VX−VY (VF=1 if no carry)
9XY0	Skip next instruction if VX=VY
AMMM	Set index = $MMM
BMMM	Goto $MMM + V0
CXKK	Set VX = (random number) AND KK
DXYN	Display N bytes at VX,VY
EX9E	Skip next instruction if VX=KEYDOWN
EXA1	Skip next instruction if VX=KEYDOWN
F000	Stop (ETI-660 uses "0000")
FX07	Set VX = current timer value
FX0A	Get key in VX (wait for KEYDOWN)
FX15	Set timer = VX x 20 millisec
FX18	Beep for VX x 20 millisec
FX1E	Set index = index + VX
FX29	Set index to display LSD of VX
FX33	Store decimal equiv. VX at index
FX55	Store V0 to VX at index
FX65	Load V0 to VX from index

Figure 2. The CHIP-8 Core Language. Instructions supported by current CHIP-8 machines.

NOTE: **X & Y** refer to Chip-8 variables X & Y (VX & VY)
KK refers to a hex number 00 to FF
N refers to a hex number 0 to F
MMM refers to a 3-digit hex address

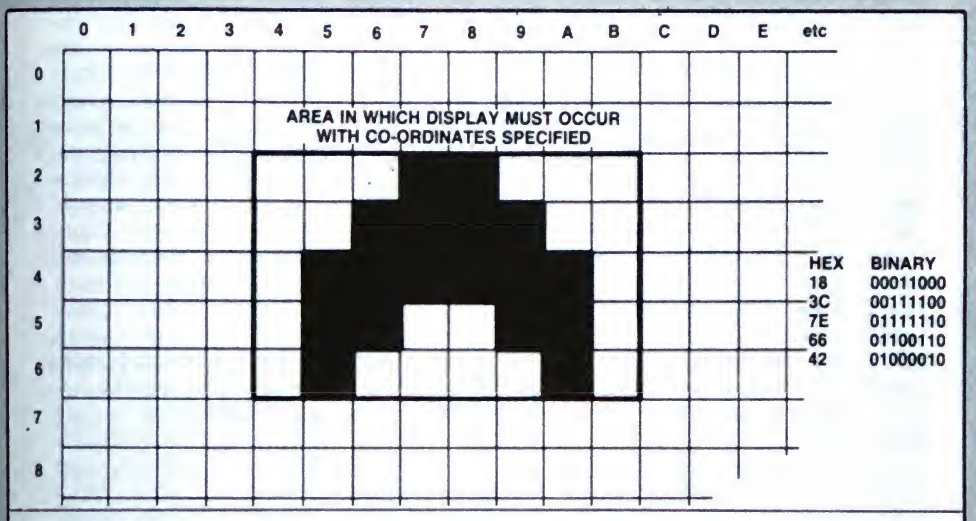


Figure 1. Creating a CHIP-8 display.

CODE	FUNCTION
8XY3	VX=VX XOR VY
8XY6	VX=VY/2 (VF = remainder)
8XY7	Output VX on Port VY
8XY8	Input VX from Port VY
F001	Toggle CHIP-8 cursor (full half-tone)
F002	Toggle display switch
F003	Change screen format
FX05	Set language execution speed = VX
FX25	Set tone frequency = VF (0 to F)
FX35	Set SYNTHA = VX (tone synthesizer function)
FX45	Set SYNTHB = VX (tone synthesizer function)
FX75	Load custom cursor VX from index
FX85	Replace cursor with custom cursor VX
FX95	Store V0 to VX in imaginary CHIP-8 screen
FXY4	Set program page ±X, data page ±Y
FXY6	Set VX,VY to display coincidence co-ordinates
FXYB	Limit joystick range to VX,VY
FXYC	Read joystick into VX,VY
FXYD	Display ASCII string at index at VX,VY

Figure 3. Extended CHIP-8 instructions. Instructions supported only by the Dreamcards interpreters.

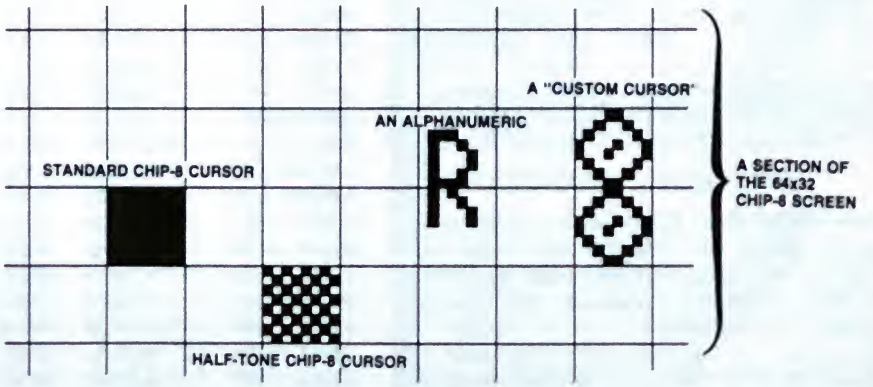


Figure 4. Cursor types available with Dreamcards' CHIP-8

Alphanumerics: Standard CHIP-8 is not a language in which you'd choose to write a wordprocessor program — its best character resolution is 16x6!!! The new interpreter makes it a good deal easier to incorporate text (such as operating instructions for a game or utility) into programs by including an instruction that will display a 'string' of standard ASCII characters in 64x16 format as if they were CHIP-8 cursors. The command even has an 'auto-repeat' facility that allows up to a whole line of any one character to be displayed using only three bytes of data.

Half-tone cursors: One difficulty with the original 'chunky' graphics was to portray contrasting tones. This can now be achieved with a simple 'switch' instruction that allows access to a 'half-tone' CHIP-8 cursor (see Figure 4) that can be used with ordinary full-tone graphics, alphanumerics and hi-res graphics. The possibilities with all these graphics combinations are limitless.

Sound effects: The Microbee tone generator is a fairly simple affair, but the new interpreter extends it to a point where it is a match for many hardware sound units. A wide range of 'buzzes', 'zaps' and 'boings' are available by manipulation of three sound commands, or they can be ignored in favour of the traditional CHIP-8 'beep'. The manual includes a listing of a routine that allows you to step through the various sound effects options — a useful feature as the number of sound effects available totals *over one million.*

Joystick: Two new instructions allow you to operate a standard Microbee joystick and to limit the range over which it is permitted to move. These functions will be invaluable to anyone who is serious about arcade games.

Extended maths and I/O: The additional exclusive-OR (XOR) and the new 'divide by 2' instructions are useful (if not Earth — shattering), but where creative programmers will really benefit is from the two new commands that allow a variable to be output to or input from a selected port. This opens a vast range of possibilities, including direct access to the ETI-673 Multiprom Board and ETI-674 Proportional Joystick, the ability to drive a printer, cassette or other peripheral device and the capacity to alter the internal workings of the PCG and colour graphics board.

Page addressing: One simple command has overcome the original 4K limit on CHIP-8 software. Programs can now be put anywhere and are limited only by the memory space available.

Conversion aids: First and foremost under this category has to be the 'switch' command that alternates between 64x32, 64x48 and 64x64 screen formats. This one command makes the *whole range* of CHIP-8 software immediately accessible to the Microbee user. In addition to this, there is an execution rate control and an instruction that writes binary data to an 'imaginary' CHIP-8 screen, both of which will greatly aid conversion of programs written for other machines in the unlikely event that difficulties are encountered. The execution rate control is also a very useful command in its own right and can be used (for instance) to simulate acceleration in a moving display.

The operating system

Without doubt the most important feature of the package is the ability of this CHIP-8 to co-exist with BASIC. A compiler converts the CHIP-8 hex code created by the programmer into BASIC 'REM' lines that can stand alone or be added to any BASIC program (see Figure 5). These can

Figure 5. A sample of compiled CHIP-8.

This is a BASIC listing created from CHIP-8 code by the Dreamcards Compiler. None of the lines were entered by hand — the *whole listing* is compiled automatically.

On typing "RUN <CR>" a decompiler converts the ASCII CHIP-8 code in lines 5 and 6 back to proper hex code at the address specified by the "header" (line 4).

Line 1 selects the ETI-673 Multiprom Board, NET#2 socket (if fitted).

Line 2 calls the "decompiler" routine in the interpreter, then executes the decompiled CHIP-8 code.

Note the checksum bytes at the end of line 4. These greatly assist program debugging.

```
00001 REM           <<<<< STRIP JACK NAKED >>>>>
00002 REM                                    by Lindsay R. Ford
00003 REM This  is  the  famous  card  game  in  which you lose
00004 REM clothes  rather  than  dollars.  The  player  and the
00005 REM computer  each  have  27  cards  and  10  articles of
00006 REM clothing.  A  coin is tossed to see who plays first -
00007 REM if  it's  "YOU"  then press a key to deal a card onto
00008 REM the  centre  deck.  The computer will then put a card
00009 REM on top of yours (and so on).  If either of you play a
00010 REM Royal card,  then  the  other player has only a fixed
00011 REM number  of  turns in which to deal another Royal card
00012 REM or  he/she  loses  the  centre  deck  and  an item of
00013 REM clothing (JACK = 1 turn, QUEEN = 2, KING = 3, ACE = 4
00014 REM and JOKER = 5). The  game is lost by the first to run
00015 REM out of either clothes or cards. Note that as the deck
00016 REM isn't  memory-mapped,  the  same  card  may  turn  up
00017 REM several times during a game.
00018 REM                    --------------
00019 REM THIS GAME  SUITS BOTH MICROBEE AND ETI 660 COMPUTERS.
00020 REM ETI 660  owners simply enter the code from $0600, not
00021 REM from $2600 as shown here (and ignore $25FE/F).
00022 REM                    --------------
00023 OUT 255,2
00024 X=USR(57347)
00025 END
00026 REM     Dreamcards Chip-8   $25F0 to 2A2F   Check 46/08
00027 REM     $25F0  0000 0000 0000 0000  0000 0000 0000 F002
00028 REM     $2600  620A 630A 641B 651B  C701 26F4 2800 00E0
00029 REM     $2610  4200 17DC 4300 17CE  6600 6C00 6A00 2700
00030 REM     $2620  7A19 3ACB 161E 26BE  26CE 26DA 26D4 26C6
00031 REM     $2630  3700 1676 8DC0 7D01  26F4 4400 17DC 27BE
00032 REM     $2640  F00A 27BE 6E19 26F6  26CE 74FF 26CE 26DA
00033 REM     $2650  7601 26DA 3601 2730  2716 3C00 1676 4D01
00034 REM     $2660  1676 3D00 1636 6E19  FE18 26BE 72FF 26BE
00035 REM     $2670  8564 6701 160A 8DC0  7D01 6E19 26F6 4500
00036 REM     $2680  17CE 27C6 26F4 27C6  6E19 26F6 26D4 75FF
00037 REM     $2690  26D4 26DA 7601 26DA  3601 2730 2716 6E04
00038 REM     $26A0  FE18 3C00 1634 4D01  1634 3D00 1678 6E19
00039 REM     $26B0  FE18 26C6 73FF 26C6  8464 6700 160A 8020
00040 REM     $26C0  6A03 6B0B 16E0 8030  6A35 6B0B 16E0 8040
00041 REM     $26D0  6A03 16DE 8050 6A35  16DE 8060 6A1C 6B1B
00042 REM     $26E0  A9B0 F033 A9B1 F165  F029 DAB5 7A04 F129
00043 REM     $26F0  DAB5 00EE 6E32 FE15  FE07 3E00 16F8 00EE
00044 REM     $2700  6B07 A9B9 DAB9 7A08  A9C2 DAB9 7A38 7B09
00045 REM     $2710  3B19 1702 00EE C001  A9B3 F055 C003 A9B4
00046 REM     $2720  F055 C00F 400F 1722  400E 1722 A9B5 F055
00047 REM     $2730  A9B5 F065 8900 A9B3  F065 4000 1742 4900
```

be EDITed, PRINTed, LOADed or SAVed exactly as if they were BASIC, but when the 'RUN' command is issued they will automatically be decompiled into CHIP-8 hex code and executed in that language.

This facility allows novice programmers to use CHIP-8 with little knowledge of how it works, whilst experts can take advantage of the combined benefits of the fast display speed of CHIP-8 and the powerful string processing and maths of BASIC within the one program.

The interpreter can be accessed from several points (detailed fully in the operating manual) that make it simple for control or data to be passed backwards and forwards between BASIC AND CHIP-8. This means that CHIP-8 subroutines can be called from BASIC programs and vice versa, a unique feature that has enormous potential for development of the language.

```
00001 OUT 255,2
00002 X=USR(57347)
00003 END
00004 REM  Dreamcards Chip-8   $2200 to 221F   Check BD/8D
00005 REM  $2200  00E0 601F 611B 6204  CA17 A21E DAB1 FA29
00006 REM  $2210  D015 F215 F207 3200  1214 D015 1206 8000
```

```
00048 REM     $2740  17AA 4900 1722 6B09  4901 1790 490D 1796
00049 REM     $2750  490C 179C 490B 17A2  A9B6 F933 A9B7 F165
00050 REM     $2760  390A 176E 6A1C F029  DAB5 7A04 1770 6A1E
00051 REM     $2770  6C00 F129 DAB5 A9B4  F065 A9E2 4001 A9E8
00052 REM     $2780  4002 A9F4 4003 A9EE  6A1C 6B10 DAB7 00EE
00053 REM     $2790  A9CB 6CFC 17A6 A9CE  6CFD 17A6 A9D3 6CFE
00054 REM     $27A0  17A6 A9D8 6CFF 6A1E  1774 A9D8 6A1A 6B08
00055 REM     $27B0  DABA 7A08 7B0A A9CE  DAB5 6CFB 00EE 6A01
00056 REM     $27C0  A9FB 2A1E 00EE 6A34  AA09 2A1E 00EE 26F4
00057 REM     $27D0  00E0 6A10 6B0C A9FB  2A20 17E8 26F4 00E0
00058 REM     $27E0  6A14 6B0C AA04 2A28  AA13 2A20 6000 6E03
00059 REM     $27F0  FE18 6E03 26F6 7001  3010 17EE 2804 1600
00060 REM     $2800  3700 00EE 00E0 2862  26F4 4500 183C 4300
00061 REM     $2810  288A 4301 289C 4302  28A6 4303 28B0 4304
00062 REM     $2820  28BA 4305 28C6 4306  28CE 4307 28D6 4308
00063 REM     $2830  28DE 4309 28E6 6E96  26F6 00EE 288A 1836
00064 REM     $2840  6002 DAB2 185C 6003  DAB3 185C 6005 DAB5
00065 REM     $2850  185C 6009 DAB9 185C  600F DABF 7A08 F01E
00066 REM     $2860  00EE 6A18 6B00 A8F2  2840 2840 6A10 6B09
00067 REM     $2870  284C 6B02 2858 2858  6B09 284C 6A10 6B1E
00068 REM     $2880  2840 6B11 2858 2858  00EE A93E 6A10 6B09
00069 REM     $2890  284C 6B0A 2858 2858  6B09 284C A966 6A1B
00070 REM     $28A0  6B0D 2846 2846 A96C  6A18 6B12 2852 2852
00071 REM     $28B0  A97E 6A1B 6B0B 2852  2840 A989 6A13 6B0A
00072 REM     $28C0  2840 2852 2840 A996  6A18 6B1D 2846 A999
00073 REM     $28D0  6A22 6B1D 2846 A99C  6A12 6B0B 2840 A99C
00074 REM     $28E0  6A2B 6B0B 2840 A99E  6A18 6B00 2840 2840
00075 REM     $28F0  00EE 0F3C E078 207F  F77F 2008 0A08 090C
00076 REM     $2900  1E0E 04FC FEFF 3E3F  3E3F 20A0 2020 60F0
00077 REM     $2910  E040 7FFF FFF8 F8F8  F808 FCDE FC08 0101
00078 REM     $2920  1E11 3F7F 7F7F 7F7F  7F7F 2424 FCFC F4F0
00079 REM     $2930  10F8 FCFC FCFC FCFC  FC48 487E 7F5F 207F
00080 REM     $2940  C07F 2080 40B0 7070  7060 4040 3413 1008
00081 REM     $2950  0100 0304 1B1C 1C1C  0C04 0458 9010 2000
00082 REM     $2960  0008 FC06 FC08 EBAA  E380 8080 162F 5F40
00083 REM     $2970  505C 5B5B 5BD0 E8F4  0414 74B4 B4B4 41E3
00084 REM     $2980  0000 081C 7F00 0000  8000 3F22 B61C 0000
00085 REM     $2990  0800 0877 00FE 38F8  08E0 F880 C000 0830
00086 REM     $29A0  2018 0000 0000 0000  0000 0000 0000 0000
00087 REM     $29B0  0000 0501 0309 0000  09FF FFFF FFFF FFFF
00088 REM     $29C0  FFFF F8F8 F8F8 F8F8  F8F8 F8E0 A0E0 A0A0
00089 REM     $29D0  C0A0 A0E0 A0A0 E040  2020 20A0 E00E 0A0A
00090 REM     $29E0  0A0E 44EE FE7C 7C38  1038 7CFE 7C38 1038
00091 REM     $29F0  7CFE FED6 3838 FEFE  FE10 38AE AAEA 4A4E
00092 REM     $2A00  A0A0 A0A0 E040 4040  E0FB AAAB AAAB 8000
00093 REM     $2A10  0000 80AB A9A9 A9FB  B929 2928 A900 6B00
00094 REM     $2A20  6005 DAB5 7A08 F01E  DAB5 7A08 00EE 0000
```

Presentation

The language and operating system comes in a 4K EPROM that fits into the 'NET' or 'Terminal' socket of the Microbee coreboard or in the 'NET#2' socket of the ETI-673 Multiprom Board. A standard 2532 type EPROM is used, meaning that owners of 'Bees purchased after April 1984 should check that their unit isn't fitted with a 2732 EPROM in the 'Telcom' ('NET') socket. If so, a free changeover ROM is available 'priority paid' from Dreamcards (details on what to look for and how to swap over are contained in a broadsheet included with the operating manual).

The operating manual itself is very detailed and contains a wealth of material that takes you from the very fundamentals of 'memory', 'hex' and 'logic' up to the use of the operating system and the extended language subsets in an easy 'tutorial' style. This wealth of information makes it about the most comprehensive manual available for CHIP-8 and a very valuable reference for the experienced programmer.

At the other end of the scale, the newcomer to CHIP-8 will rapidly become familiar with the language by sitting down at the keyboard and carrying out the program examples.

In addition to a large number of demonstration programs designed to illustrate important features of the various instructions, the manual also contains source listings of two very useful machine code utilities and a full disassembler program. ■

THE CHIP-8 INTERPRETER/ COMPILER EPROM

is available from:

Dreamcards
8 Highland Court
Eltham North 3095 Vic.

It is only available by mail order from this address. It may also be obtainable from your local Dreamcards software dealer. It costs $49.95 (rrp) plus $1 post and handling.

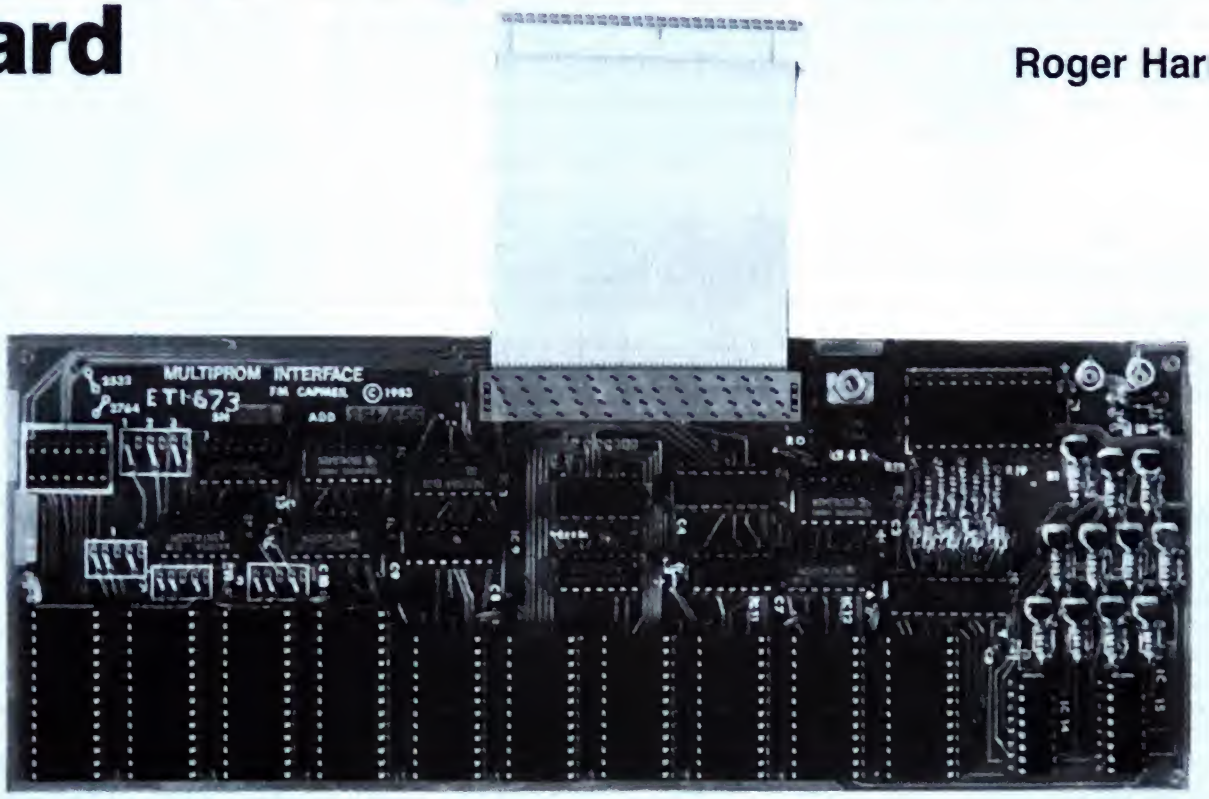
CHIP-8 USERS' NEWSLETTER

An informative CHIP-8 user group newsletter, called "Dreamer", is published by **Frank Rees of 27 King St, Boort 3537 Vic.** You can get a 10-month sub. (September '84 to June '85) for $15, or 16 months (to December '85) for $24. Each issue contains articles on hardware additions and modifications to CHIP-8 computers plus software, tutorials, etc.

Project 673

Multiply your Microbee's ROM capacity with this interface board

Roger Harrison



This project design by F. M. Capmeil, presented with the cooperation of Avtek Electronics, permits extending the ROM capacity of the Microbee to 44K, or to 308K total by daisy-chaining up to six slave boards to produce a sort-of high speed 'read-only virtual disk' (a ROD?). It takes 2532s or 2764s (can be mixed) and, in addition, provides 11 open-collector outputs and eight buffered inputs.

THIS PROJECT gives you the ability to extend your Microbee's ROM software capacity. It is a board that just plugs into the 'Bee's 50-way expansion buss and can either be fitted inside the 'Bee or externally. Ten sockets accommodate the ROMs and either 2532s or 2764s can be used — mixed, too, if you like. The first four sockets are 28-pin types, while the last six are 24-pin types. The latter take only 2532s while the first three take either 2532s or 2764s. Socket no. 4 is arranged to take only a 2764. Address-wise, the first seven sockets are located at C000 to DFFF (i.e: where EDASM sits), while the last three are located at E000 to EFFF (NET location). Sockets 1, 2, 3 and 5, 6 7 are paired — 1:5, 2:6 and 3:7, when 2532s are in use. Only the first three are used when 2764s are in place. Link fields on the board permit ROM type selection.

In addition, eight buffered input lines are provided along with 11 open-collector transistor output lines. Using this board with special purpose-written software in ROM, you can turn your Microbee into a burglar alarm, a process controller or such like.

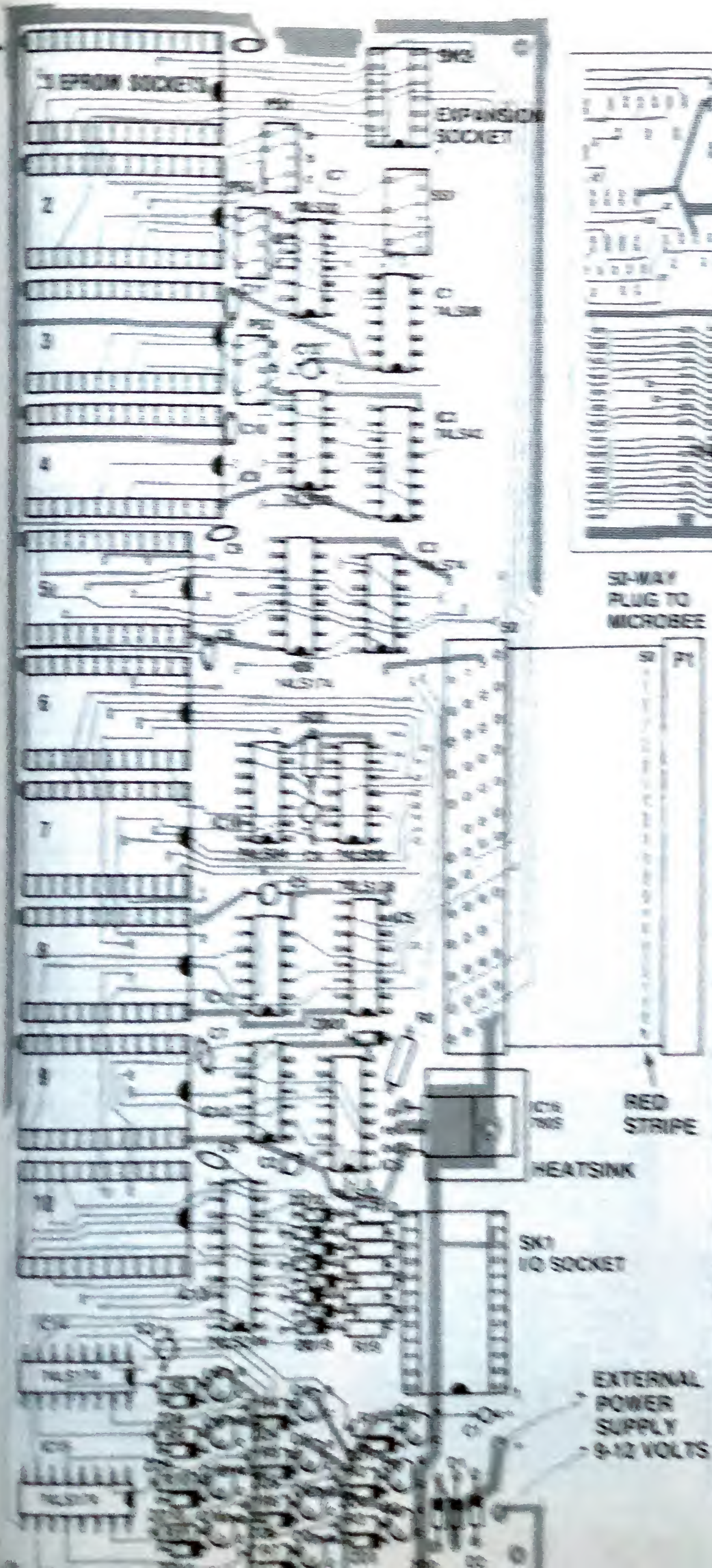
A cassette of 'driving' software is provided with kits. This contains two 'monitor' programs, one in BASIC, the other in machine code. Using the monitor you can utilise the input and output lines and, as data for these needs to be in decimal form in programs, but it is in binary form when inputting or outputting, a decimal-to-binary conversion program is included for your convenience.

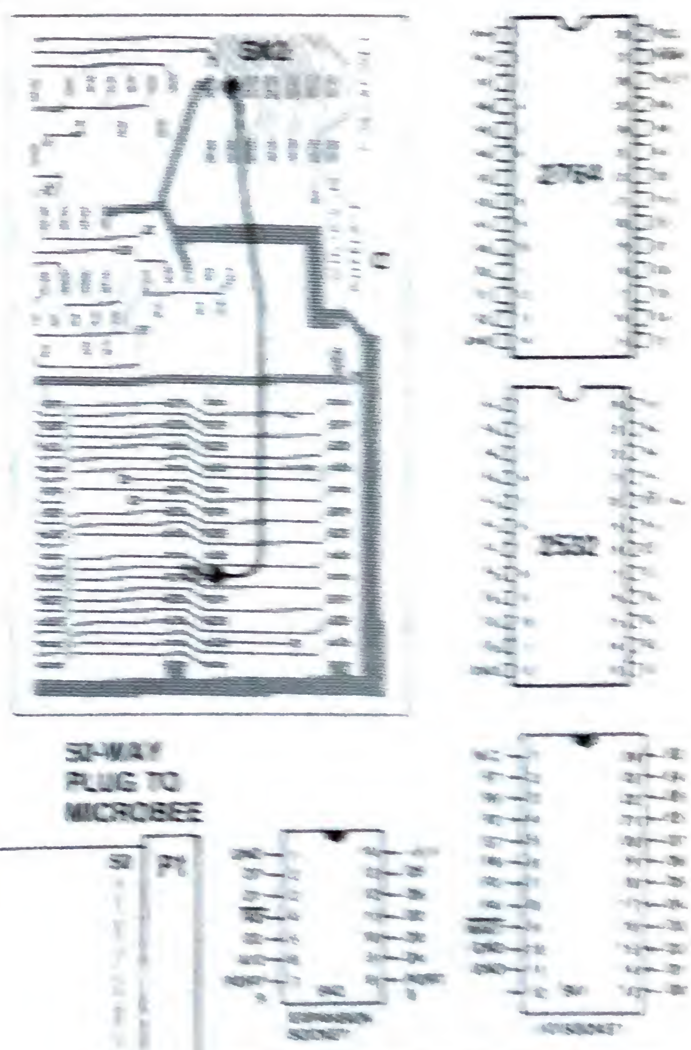
Copyright on the pc board is held by Avtek Electronics who are marketing the kit. Hence, we have not reproduced the artwork. In anycase, the board is double-sided with through-plated holes, not something most hobbyists are capable of making for themselves! To ease construction, the board is solder-masked and the component side has a silk-screened overlay showing component locations. For those who can provide components out of their own stock, Avtek will sell you a board and monitor cassette alone.

Construction

Assembling the board is quite straightforward. Sockets can be used on the fifteen DIL ICs if you wish. Commence construction by soldering all the IC sockets in place, noting orientation. Follow with the resistors, capacitors and diodes, ensuring you get the tantalum capacitors and diodes the right way round. Then solder the 11 transistors in place. The three-terminal regulator (IC16) can be assembled next. It needs a small heatsink, a Thermalloy #6073 or similar should be adequate. Insert the regulator's three legs into the board holes and then bolt it and the heatsink to the board before soldering the legs in place. Smear a little thermal compound on the rear of the regulator's flange before assembling it, to improve heat conduction to the heatsink.

Insert the ICs now, or solder them in place if you're not using sockets. *Beware you get the ICs correctly orientated.* ▶





PARTS LIST — ETI-673

Resistors .... all 1/4W, 5%
- R1, 12-19 .... 10k
- R2-11 560
- R20 .... 1k

Capacitors
- C1 .... 1µ/25 V tant.
- C2-8, 10 .... 10µ/16 V tant.
- C9-13 .... 10n ceramic

Semiconductors
- D1, D2 .... 1N4001, 1N4002
- IC1 .... 74LS08
- IC2, IC3 .... 74LS42
- IC3 .... 74LS74
- IC4 .... 74LS30
- IC5 .... 74LS138
- IC7, 8, 10 .... 74LS32
- IC9, 14, 15 .... 74LS174
- IC11 .... 74LS04
- IC12 .... 74LS32
- IC13 .... 74LS244
- IC16 7805
- Q1-11 .... BC548
- ZD1 .... 15 V, 1 W zener
- ZD2-11 .... 27 V, 400 mW zener
- ZD12-21 .... 5V6, 400 mW zener

Miscellaneous
- P1 .... 50-way I/D plug
- SK1 .... 24-pin IC socket
- SK2 .... 14-pin IC socket

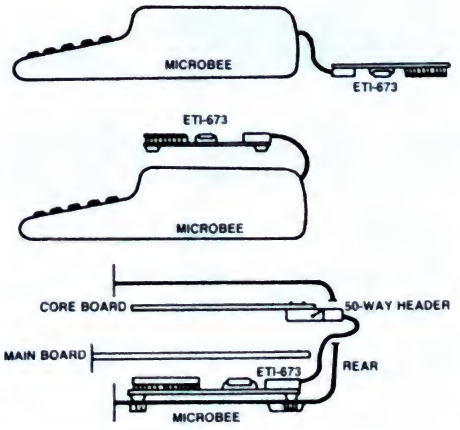
ETI-673 pc board, 50-way I/D header sockets, 50 mm of 50-way ribbon cable, IC sockets — 4 x 28-pin, 6 x 24-pin, 1 x 20-pin, 6 x 16-pin, 5 x 14-pin; heatsink — Thermalloy [illegible] or similar; 14-pin DIP I/D plug in header; 24-pin DIP I/D plug in header; ribbon cable to suit, etc.

Estimated price: $90-$110

A wire link has to be installed on the rear of the board, from pin 2 of the expansion socket (SK2) to pin 19 of EPROM socket 1.

The 50-way header-cable-socket assembly is mounted last of all. Insulation-displacement (ID) connectors were used on the prototype. The 50-way cable is grey and has one lead marked with a red stripe. This is the pin 1 line, as shown on the component overlay. Only a short length of cable should be used as the lines are unbuffered. No more than 80 mm is necessary.

Give the board a thorough check and, if all's well, you're ready to roll.



Installation

If your Microbee doesn't have an expansion socket fitted, then you'll need to obtain one and fit it. A male 50-way right-angle pc mount type is needed, the same as is used with disk drive interfaces. It mounts under the Microbee's core board. If your 'Bee is still under warranty, your supplier can probably fit one for you.

The MultiPROM board can be mounted in three ways, as shown in the accompanying diagram: on top of the Microbee case, upside down at the rear or inside of the Microbee under the main pc board. The choice is up to you. ▶

HOW IT WORKS — ETI-673

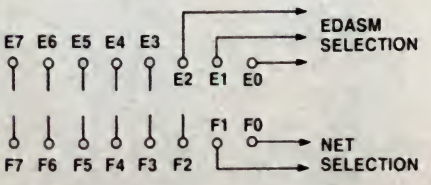
A total of ten EPROM sockets have been provided. Sockets 1, 2 and 3 can accept either type 2532 or 2764 EPROMs, while socket 4 only takes a 2764. Sockets 5 through 10 take only 2532s. Pin numbering on the 28-pin types (for 2764s) is shown in brackets. Sockets 1, 2 and 3 are paired with sockets 5, 6 and 7 respectively, but only sockets 1, 2 and 3 will accept 2764s (5, 6 and 7 then being unnecessary when 2764s are in place). On-board link fields permit selection between ROM types. These fields are labelled SS1, PS1, PS2 and PS3. Use of the link fields is explained in the text.

The 'output enable' line (OE) of the selected ROM, or ROM set, is driven low to access it, selected by the address and data buss decoding scheme that is part of the output selection circuitry. Sockets 1 through 7 are located at C000 to DFFF in the 'Bee's memory map, known as the EDASM (editor-assembler) location. Sockets 8, 9 and 10 are located at E000 to EFFF, the NET (network) location.

The project uses two of the Z80 I/O ports for both EPROM selection and the input/output control. These ports are 254 (FE hex) and 255 (FFF hex).

The output port 255 has eight outputs labelled F0 to F7. Three of these outputs F0, F1 and F2, are used to select the EPROMS in the EDASM location (sockets 1 through 7).

The output port 254 has eight outputs labelled E0 to E7. Two of these outputs, E0, and E1 are used to select the EPROMS in the NET location.



The remaining outputs are used to drive 11 transistors. They are F3 to F7 and E2 to E7. The collectors of the transistors are brought to the input/output socket, SK1.

The A1 to A7 Z80 address buss lines from the 'Bee are gated with an inverted copy of the I/O Request (IORQ) line. When on-board ROM is to be selected, the A0-A7 address lines will be high and IORQ low. The A0 line determines which Z80 port is selected, being low when port 254 is selected, high when 255 is selected. The RD line will be low.

A data value on the data buss is latched by IC9 from the lower three lines, D0-D2, and outputs E0, E1 and E2 are decoded by IC2. Similarly, IC3 latches the value on the data buss and IC6 decodes outputs F0 and F1.

The ROM selection signals are gated with the appropriate output from IC5 which decodes the upper four address lines, A12-A15. Software in the selected EPROM will only be read when both the MERQ and RD lines are low and a valid address is present.

The selection of the various EPROM sockets is effected by the output lines as indicated in Table 1.

F2	F1	F0		E1	E0	
0	0	1	EDASM 1	0	1	NET 1
0	1	0	EDASM 2	1	0	NET 2
0	1	1	EDASM 3	1	1	NET 3
1	0	0	EDASM 4			

Table 1. EPROM selection code

It should be evident that every time a port is written to, the output has to take into account which EPROM is meant to be selected. This may seem a little complicated but it allows complete software control on the board. The EPROM selection can be changed within execution of a program, allowing complex jumps between programs residing in different EPROMS located at the same address. This effectively extends the available memory in ROM from 12K to 44K per board, with 12K available at a time.

The output responds to the basic command:

OUT port no data

example: OUT 255,47

(both values are in decimal notation).

In the command: OUT 255,2 — 2 represents the data. This data will be translated into an eight-digit number in binary, in this case: 00000010. Each of the digits will be put onto one of the output lines, as follows.

F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	1	0

A "1" means that a voltage is applied to the transistor driver and it will turn on. A zero means that no voltage will be applied to the transistor and it will not conduct.

If you want to turn on one single output line it is necessary to know the decimal equivalent of the complete 8-digit binary number formed by the eight outputs, before that particular line can be driven high. The cassette monitor software includes a binary-to-decimal conversion routine for your convenience.

There are two ways to write to the output: (1) The decimal value corresponding to the output required is known. In this case just output to the desired port. (2) Only one data line needs to be changed from last output, in which case it is necessary to hold the binary value in an array, change the bit corresponding to the line to alter, convert to decimal, and output. In this way one could write alternately to several output ports after setting or resetting the desired lines.

The input is connected to both port 255 and 254. Reading from port 255 is the same as reading from port 254. The input port has eight inputs labelled from D0 to D7 (not to be confused with the Z80 data buss). If the board uses a different address pair the input will also respond to both.

The input responds to the BASIC command:

variable = IN (port no.)

example: A=IN (255), where the port address is in decimal.

IC13 buffers the input lines which are protected against excess voltage or reverse polarity by the 100 ohm series resistors and 5V6 zeners across each line.

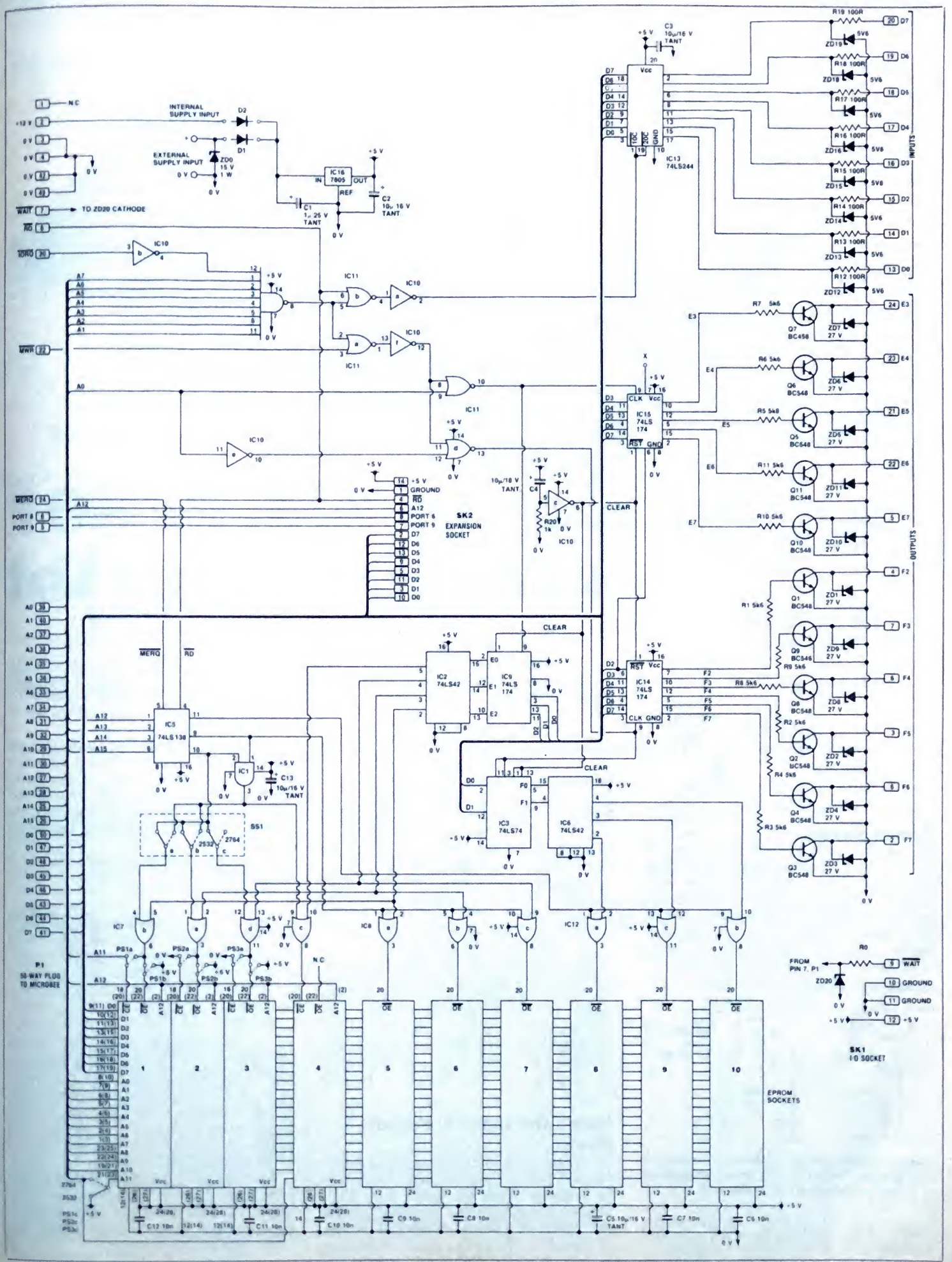
The user outputs, F2-F7 and E3-E7, comprise a series of open-collector BC548s, Q1 to Q11. Each transistor is protected against supply over-voltage and reverse polarity by 27 volt zeners. The E-series outputs are driven by IC15, the F-series by IC14. These two ICs latch the data value from the Z80 address buss, their outputs driving the bases of the output transistors.

Power-on 'clear' is effected by IC10c. When first powered-up, C4 will be discharged and the input of IC10c will be driven high. After C4 charges, the input of IC10c will go low, clearing all the outputs of ICs 14 and 15. The output of IC10c also clears latches IC6 and IC9 and the EPROM selection circuitry.

Supply input can be between nine and 12 V dc. IC16 is a three-terminal regulator that provides the +5 V supply for the board. Capacitor C1 maintains the stability of IC16 and C2 provides output bypassing. Diodes D1 and D2 provide mutual supply rail blocking while ZD0 protects against over-voltage and reverse polarity input on the external supply input.

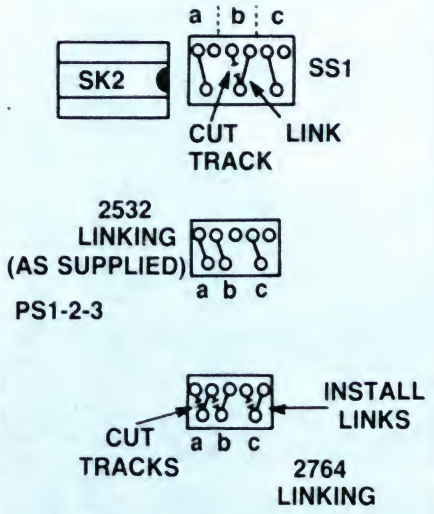
The expansion socket SK2 brings out the Z80 data buss plus 'Bee expansion connector signals RD, address line A12 and ports 8 and 9, along with +5 V and ground.

Note that there is a Z80 WAIT signal input on the I/O socket, SK1 (pin 9). Protection is afforded by R0 and ZD20.


INTERNAL SUPPLY INPUT
EXTERNAL SUPPLY INPUT
TO ZD20 CATHODE
SK2 EXPANSION SOCKET
SK1 I/O SOCKET
EPROM SOCKETS
P1 50-WAY PLUG TO MICROBEE
INPUTS
OUTPUTS
CLEAR
IC13 74LS244
IC15 74LS 174
IC14 74LS 174
IC5 74LS138
IC2 74LS42
IC9 74LS 174
IC3 74LS74
IC6 74LS42
IC16 7805

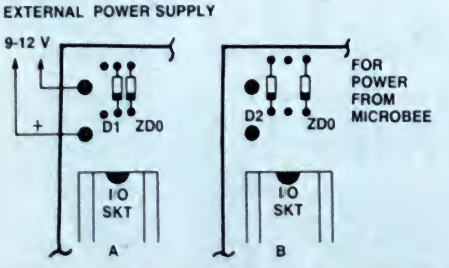
Before using the board you'll need to decide what EPROM types you'll be using and set up the linking on SS1, PS1, PS2 and PS3 accordingly. As it comes, the board is set up for 2532s with tracks linking the connection pads in the field areas on the top side of the board. The appropriate tracks must be cut and links installed as shown here.

PROM type selection. The first three sockets can accept either 2532s or 2764s. Four link fields permit individual selection. As it comes, the board is set up for 2532s. Link field SS1 permits you to designate the EPROM type — a for skt 1, b for skt 2, c for skt 3. For 2764s, cut the existing track and install a link to the right, as shown. The A11, CE and OE pins for each socket also have to be jumpered when 2764s are selected. Pin selection fields PS1, PS2 and PS3 effect this. They're located near each socket. Again, cut the existing track and jumper to the right, as shown, on the pin selection field for the appropriate socket.



Power supply

The project can be powered from the Microbee or from an external power supply. Microbees have been supplied with different power pack models at different times since first released. Some are able to drive both the 'Bee and the MultiPROM board, some can't. The sign of a power supply with insufficient 'grunt' in this application is: RESET doesn't operate!



Internal/external supply. Connections for using an external power supply (A); (B) shows arrangement for obtaining power from the Microbee.

The board can be powered externally from a small 9 Vdc power pack capable of supplying 200 mA or more (current depends on the number of ROMs inserted). The Dick Smith M-9560 9 Vdc/600 mA plugpack or similar would be fine.

Input/output

The project uses two of the Z80 I/O ports for both the EPROM selection and the input/output control. The ports used are:

port 255 (hex FF)
port 254 (hex FE)

The ports used can be changed by inverting one of the address lines via a spare inverter provided on-board. This function allows the connection of several boards all at a different address. The pairs of addresses available are:

255/254
253/252
251/250
247/246
239/238
223/222
193/192
127/126

Software selection

The EPROMs are selected by an OUT statement from BASIC. The address of the EDASM location is 255. To select the first EPROM of the EDASM location set, enter:

OUT 254,1

then hit the *return* key. Then enter

EDASM

followed by the *return* key.

As another example, if you wish to select the third EPROM in the NET location set, enter:

OUT 255,3

then hit the *return* key and enter

NET

followed by the *return* key, again.

A cold start will not change your EPROM selection but a power OFF will. If your 'Bee has a battery backup, the data will not be lost but before anything, you must reselect the proper EPROM.

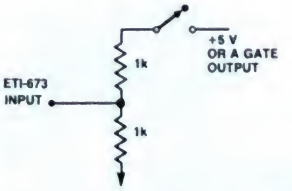
Example: OUT 254,1 (Wordbee), type EDASM. You are in Wordbee. Then enter text. Turn power OFF. Turn power on, the screen should display

OPTION NOT FITTED

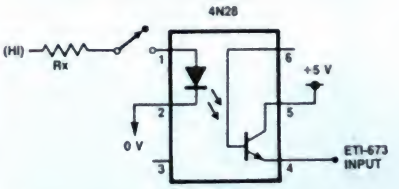
This is because the 'Bee does not see any EPROM until they are selected. So, enter OUT 254,1 (Wordbee), type EDASM. You should be back in Wordbee. If your 'Bee appears to hang, then reset. It is a good idea to always leave the monitor when turning the power off as this leaves a return address in memory and will prevent a possible hangup.

Using the input & output lines

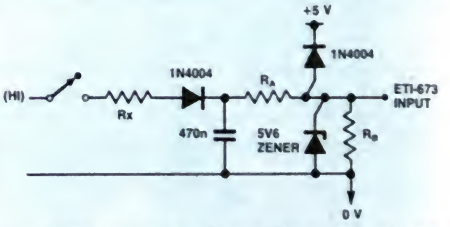
To use the input and output lines on-board, you will have to interface them to some sort of hardware. Take the inputs first. There are two forms the input can take: reading the 'status' of a switch, which can be open or closed; or the input can be driven from the output of a logic gate. To read the status of a switch, use this circuit:



However, this is only useful where the switch is less than a few metres away. Where the switch or gate is at the end of a long line, use an optocoupler at the MultiPROM input, like so:



If the input is a pulse or varying dc level signal, then it is necessary to use a signal conditioner. This circuit should do the job:

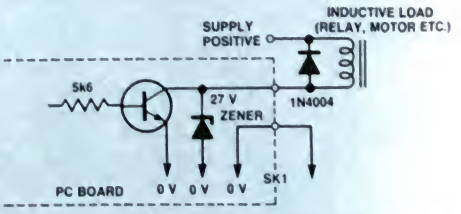


To drive an input direct from a logic gate output, no special interface is required.

In all cases, use twin-pair cables; one to the input, the other to 0 V.

Now for the outputs. The output lines are 'open' BC548 collectors. That is, an output device, such as a relay, is connected between a power supply and the collector. When the output is programmed ON, the transistor conducts, operating the output device.

It is a simple matter to drive a relay or dc motor. Like so:



The maximum supply voltage the BC548s will stand is 30 V, so any supply should be kept below 90% of this value (i.e: below 27 V). A 27 V zener is used to protect each output transistor. note that any external supply used to power the device being switched should have its 0 V line returned to the MultiPROM's 0 V line.

We don't have the space here to go into programming details, so that will have to be left to the kits; these are supplied with a manual which includes programming and other software information. ■

16K to 32K expansion for your MicroBee

Considering going on to bigger and better things with your MicroBee? Not afraid of a little soldering and want to save money? Here's how!

Tom Moffat

PAGE TWO of my MicroBee construction manual says: "Should you wish to upgrade your 16K MICROBEE you will need to purchase a 32K changeover CORE BOARD. WARNING do not attempt to upgrade your 16K board as this will invalidate your warranty and will require an updated BASIC anyhow."

But don't believe everything you read. It is entirely feasible to upgrade your own MicroBee memory, if you take care. Yes, you will blow your warranty, but the warranty only runs for 90 days. That much time has probably passed already now that you've learned to use your MicroBee, and want to move on to bigger and better things.

As for the "updated BASIC" all I can say is the BASIC in my MicroBee handles expanded memory nicely without any updating. It took a bit of digging to discover this. The MicroBee Users Manual gives the 'top of memory' pointer location as 00A0 (hex). The 'Search' feature of the MicroBee Monitor was used to find a load instruction to that address. Sure enough, there it was in the BASIC ROMs at 857D. It was then necessary to work backwards from that address, disassembling the code by hand, to find how the data for the top of the memory pointer was developed.

It seems that the MicroBee finds out where the memory ends automatically by testing each byte and declaring "end of memory" when the test fails or when the start of the BASIC ROMs is reached. So it can cope with any amount of memory up to 32K, in any sized blocks. The relevant part of BASIC is reproduced in Listing 1, for those interested.

Pros and cons

Now to the pros and cons of doing it yourself. The prime advantage is financial . . . it cost exactly $75.40 to turn my MicroBee 16 into a MicroBee 32. Applied Technology charges $125 to change a 16 into a 32 PLUS. The 'PLUS' part involves supplying the parallel port plug and the battery backup arrangement, another seven dollars or so worth of goodies.

This is not to suggest that Applied Technology is ripping anybody off. If you took your 'Bee to your local electronics shop they'd probably charge more than $125 for the job. After all, there's a lot of labour in it. If you do it yourself, you supply the labour and you save the shipping which, from Tasmania at least, is dreadfully expensive.

Now before you get stuck into it, it's time to do a bit of soul-searching. Are you technically competent to do it? Is your soldering up to truly *professional* standard? Is your soldering *iron* up to professional standard? Something like a Weller or Adcola temperature-controlled soldering station is needed with the finest possible tip.

Do you have one, or can you borrow one? Look at the pictures, you'll notice that just about all IC pins have other tracks running between them on the board. It will be a very delicate job, and a mistake could ruin your MicroBee forever. If you don't feel you're up to it there's certainly no shame in handing it over to Applied Technology.

Ready?

If, after all that scary stuff, you still want to press on, here's what to do. First, get your ICs and eight 10n 'blue chip' ceramic bypass capacitors, or similar. My ICs were bought from a normal supplier in Hobart, and the capacitors were ratted from a defunct computer board (these are usually very good quality).

Because of the MicroBee's memory testing feature it is possible to upgrade your computer in 2K stages. You only install the chips you need, or can afford. Table 1 sets out what chips are required for each level of 'K', and it should be understood that each level's ▶

requirements are in addition to previous levels. Even if you're going for the full 32K it's a good idea to upgrade 2K at a time, testing as you go.

To begin the job, open up the MicroBee, disconnect the memory battery (if installed), and carefully unplug the core board. If you don't know what the core board is STOP right now and don't attempt the upgrade.

Put the computer aside and lay the core board on the workbench. Study the silk screened labelling on the top (component side) and work out what ICs go where, and which way around. In other words think the job through in advance.

Now, install the eight bypass capacitors in their positions next to the empty memory chip locations. You might as well put them all in even if you're not installing all the memory now.

Next come the ICs. Start with the small ones first to get a bit of practise for the big ones. When soldering them in, remember that they are CMOS devices and you should take the usual precautions against static damage. Don't handle the pins, pick up the packages with your thumb and forefinger holding the ends only. Use a soldering iron with an earthed tip and solder the supply and earth pins first.

TOTAL #	74LS138	4053	6116P	PEEK(161)
18	IC14	IC16	IC13	72
20			IC15	80
22			IC17	88
24		IC19	IC18	96
26			IC20	104
28			IC21	112
30		IC23	IC22	120
32			IC24	128

Look at the close-up photograph. Notice that the soldering iron tip is applied to the pins from the 'inside the IC' direction. You press down on the pad and the pin at the same time and then apply the solder from the 'outer' side of the pin. Each pin will take about two seconds to do, keeping the total heat input to a reasonable level.

The results should look like the photograph . . . just enough solder to do the job and no more.

LISTING 1

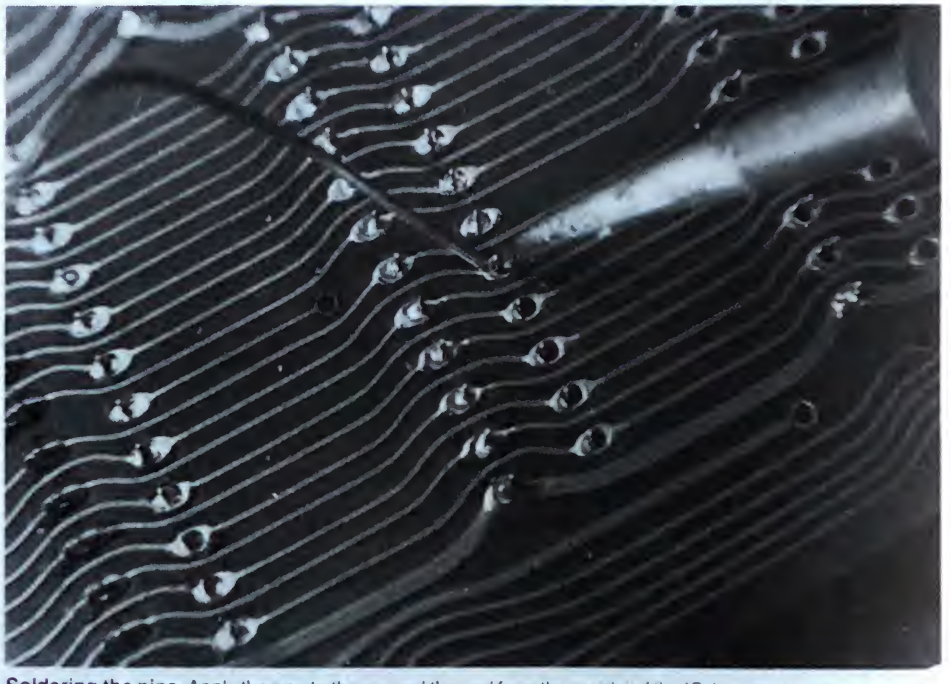
```
ADDR   CODE     LINE   LABEL  MNEM    OPERAND

                00100 ;END OF MEMORY DETERMINATION, FROM BASIC. 16/1/83
                00110
0400            00120          DEFR    16
8567            00130          ORG     8567
E000            00140 DEBUG    EQU     0E000
8567 210009     00150          LD      HL,0900 ;START OF BASIC
856A 3E80       00160 JUMP1    LD      A,80
856C BC         00170          CP      H
856D 280E       00180          JR      Z,JUMP2 ;GET OUT IF HL=8000
856F 7E         00190          LD      A,(HL)  ;LOAD BYTE INTO ACCUMULATOR,
8570 2F         00200          CPL             ;TURN IT UPSIDE DOWN,
8571 77         00210          LD      (HL),A  ;AND PUT IT BACK.
8572 2B         00220          DEC     HL
8573 70         00230          LD      (HL),B  ;PREVIOUS BYTE INTO B
8574 23         00240          INC     HL
8575 BE         00250          CP      (HL)    ;COMPARE A WITH (HL)
8576 2005       00260          JR      NZ,JUMP2;NO GOOD? GET OUT.
8578 2F         00270          CPL             ;TURN BYTE OVER AGAIN
8579 47         00280          LD      B,A     ;AND PUT IT INTO B.
857A 23         00290          INC     HL      ;NOW DO NEXT BYTE.
857B 18ED       00300          JR      JUMP1
857D 22A000     00310 JUMP2    LD      (0A0),HL;TOP OF MEMORY POINTER
0000            00320          END
00000 Total errors

JUMP2   857D    JUMP1   856A    DEBUG   E000
```

Note

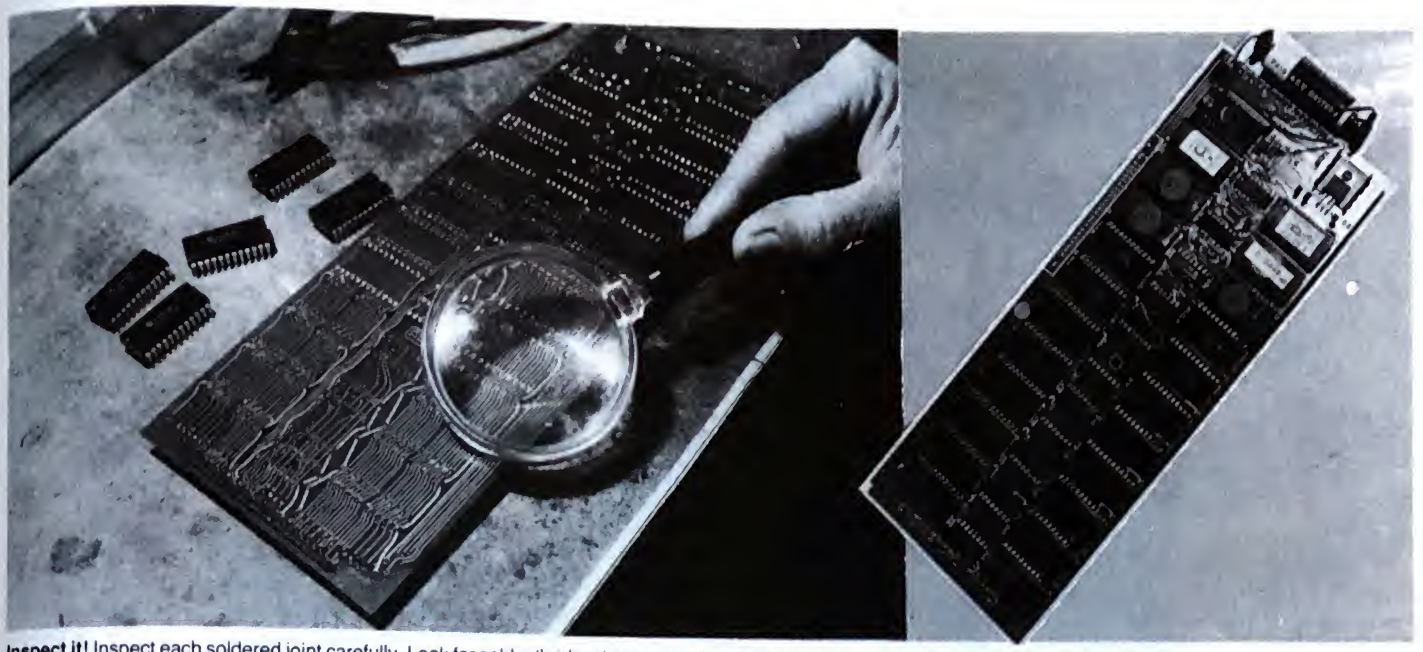
Following the first publication of this article, a reader, J. Richards of Jamboree Heights in Queensland wrote:

Microbee owners who attempt this suggestion for upgrading their machines from 16K to 32K should be aware of a problem they will encounter as a result of not having the revised BASIC ROM set supplied by Applied Technology.

The membry test routine as listed will leave the last byte of memory complemented if RAM is found to extend to address 7FFFH. Routines that store data extending from the top of RAM downwards will find this data corrupted on each reset or self-test.



Soldering the pins. Apply the iron to the pin and the pad from the 'inside' of the IC, heat the joint, *then* apply the solder. Use only enough to do the job — and no more.



Inspect it! Inspect each soldered joint carefully. Look for solder 'bridges' across tracks, pins not soldered, etc.

Finito! The completed coreboard — now the 'full bottle' 32K.

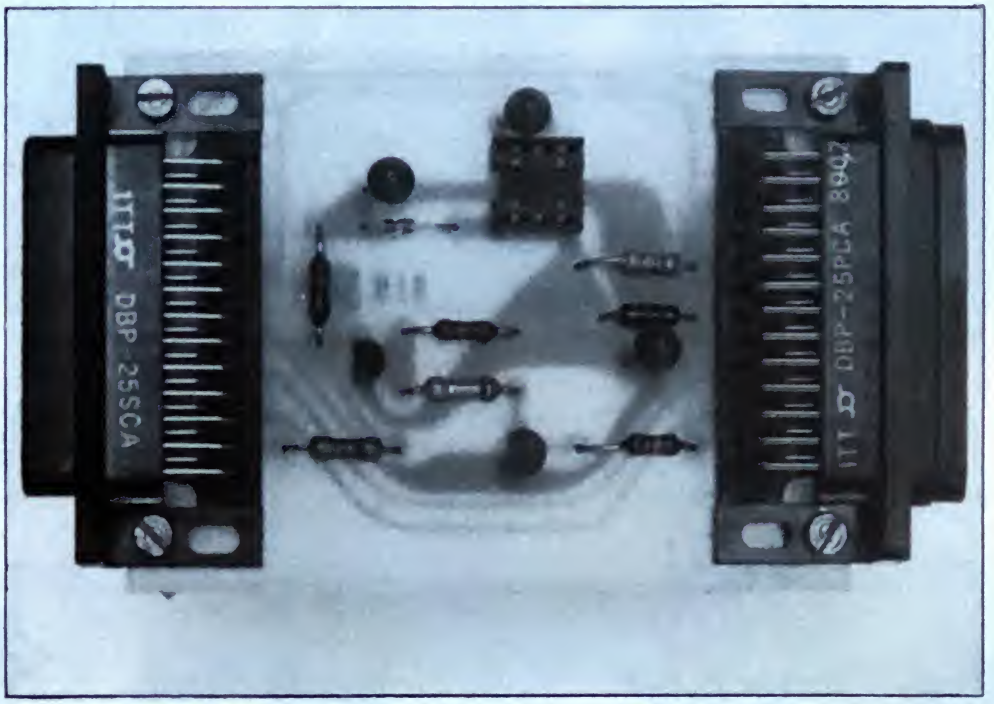
Test as you go

Testing your work is simple. After each 2K, reinstall the core board and fire up the computer. It should 'beep' into a cold start. If you inspect location 161 (decimal) with a PEEK instruction, you will find where the MicroBee thinks the end of memory is. If it agrees with where *you* think the end of memory is, congratulations.

If you've gone for the full 32K, try out the 'self-test' routine by holding down 'S' and operating RESET at the same time. The computer should now pass the '32K RAM' part of the test that it had always failed in the past!

■

A 'fair dinkum RS232er' for the Microbee



The Microbee, among other home computers, has a 'sort of' RS232 port in that it doesn't implement the negative-going portion of its output signal (TxD). Most peripherals with an RS232 input can cope with that, but inevitably, there are those that can't — as Bob found out. This project fixes that.

Bob Martindale VK3BMA

HAVING HAD my Microbee for over 12 months, and after playing the usual games, etc, it came time for some 'serious' work for the machine. At about this time, I gained access to a high quality daisy-wheel type printer — a Diablo 1650 word processor terminal (*very* smart!) — which, fortuitously, is provided with a 1200 baud RS232 interface socket. Bewdy! I thought, and proceded to make arrangements to obtain super quality listings of my programs. After checking port pinouts, a patch cable was assembled and the system fired-up. It didn't work!

Application of an oscilloscope indicated that the Microbee's TxD output signal was switching between 0 V and about +10 V. Reference to the 'Bee's circuit diagram revealed a transistor switching stage powered from the +12 V (nominal) supply rail. A quick check of Graham Wideman's article *Beating the RS232 Blues* in ETI for August 1982 indicated there should also be a negative signal excursion of between 3 V and 12 V amplitude if, in fact, the Microbee's output was to be 'true' RS323. Hmmm ... could that be the culprit?

I quickly assembled a switching adaptor (two transistors) powered from positive and negative supply rails on solderless breadboard and tried the system again. It worked first up, and voila! — super quality printouts.

I then reassembled the circuit on a small piece of Veroboard, adding a negative supply rail inverter (Intersil ICL7660) to make the unit self-contained and capable of being powered from the Microbee's supply rail (available on pin 9 of the RS232 port). I managed to make it small enough to mount, out of sight, inside the backshell of a DB25 plug. However, ETI has made a pc board version that simply 'inserts' between the 'Bee's RS232 port and the peripheral's RS232 plug.

Construction

Assembly diagrams are given for both the Veroboard and pc board versions. A track-cutting diagram is given for the Veroboard. This should be done first if

PARTS LIST — ETI-676

Resistors .....all ¼W, 5%
- R1 .....220R
- R2 .....220k
- R3 .....22k
- R4 .....100k
- R5 .....3k3
- R6 .....1k

Capacitors
- C1, 2, 3 .....10μ/10 V tant.

Semiconductors
- D1 .....1N914, 1N4002, etc
- IC2 .....ICL7660
- Q1 .....BC549
- Q2 .....BC558
- ZD1 .....6V8 zener

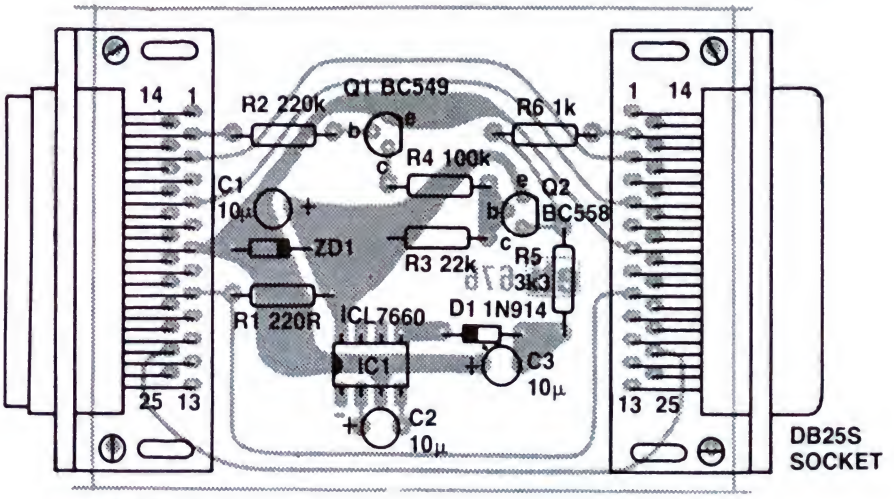
Miscellaneous
ETI-676 pc board; DB25P right-angle pc mount plug; DB25S right-angle pc mount socket; 4 x 4BA bolts, nuts and washers.

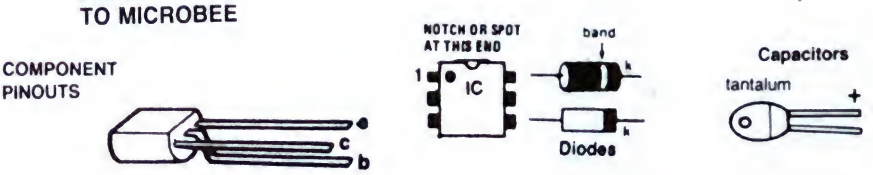
Price estimate $32-35

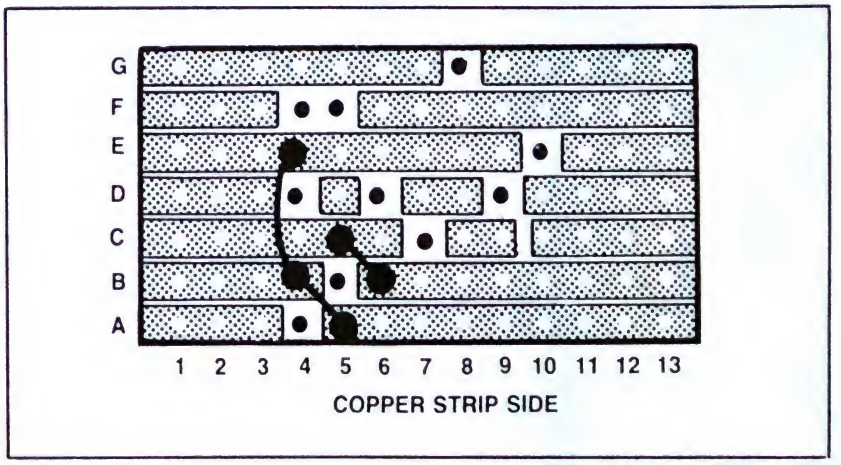
you're assembling the project this way. Note that two links are required, but these should be soldered in later. An IC socket may be used for ICI but note that pins 1, 6 and 7 *do not* make connection to the copper strips of the Veroboard. If you're using a socket, snip off or remove pins 1, 6 and 7, otherwise cut off the pins from the IC.

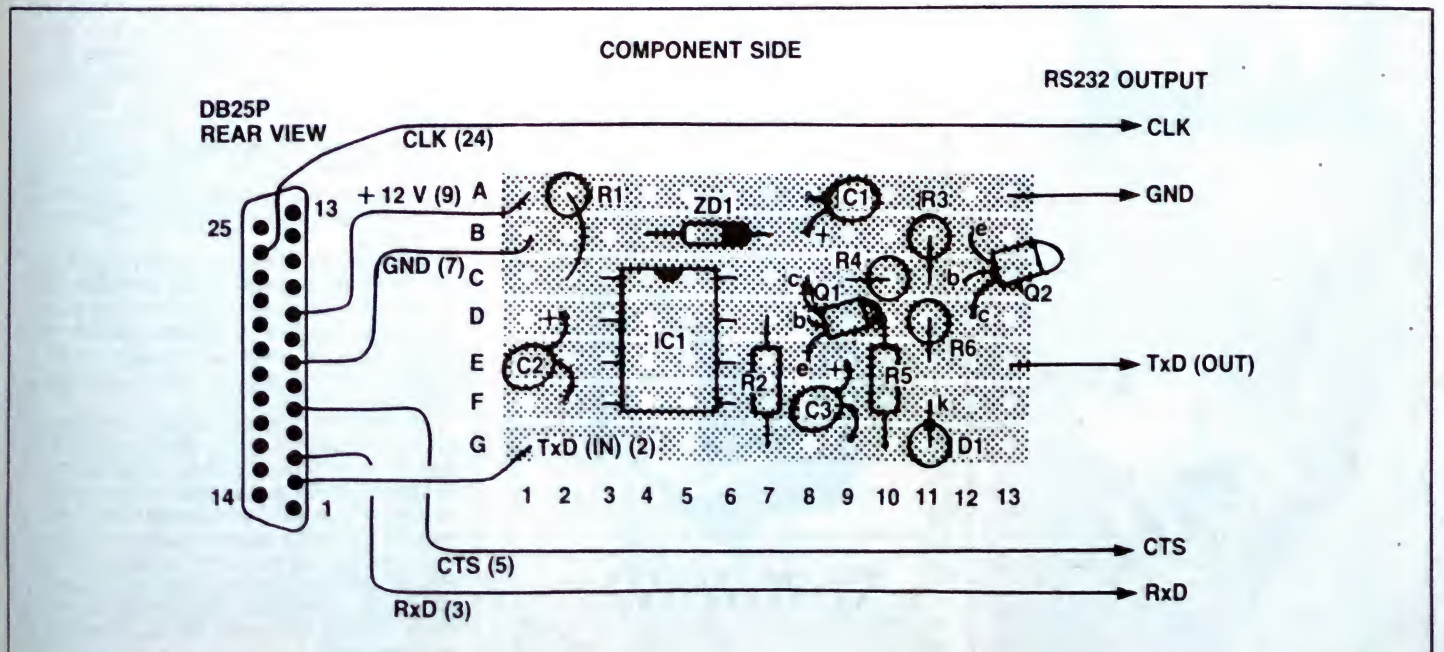
The components can be assembled in any order, but, as usual, watch orientation of the semiconductors and tantalum capacitors. Note that some components are stood on end. Seat the components right down on the board and you'll fit the project in a DB25 backshell without too much difficulty. Wire the assembly to the DB25 plug with short lengths of hookup wire and wire the cable to TxD and GND. Note that RxD, CTS and CLK should be wired straight through to the DB25 plug.

The pc board version is also simple to assemble. Just follow the overlay. Carefully check the pc board before assembly. ▶



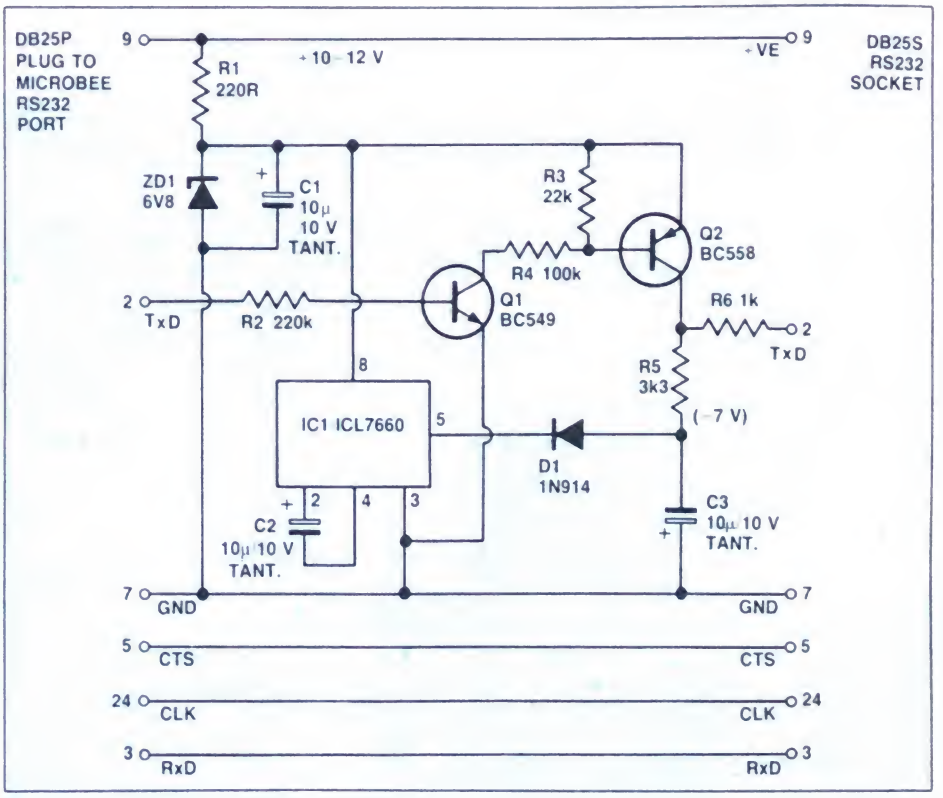


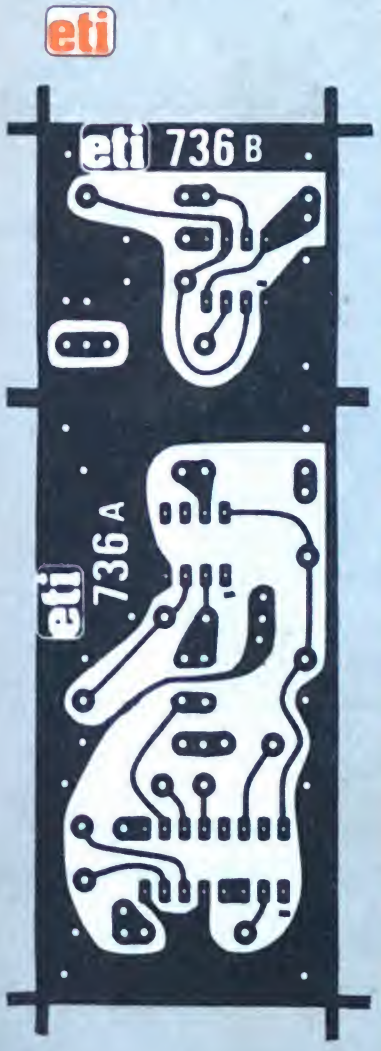




PCB

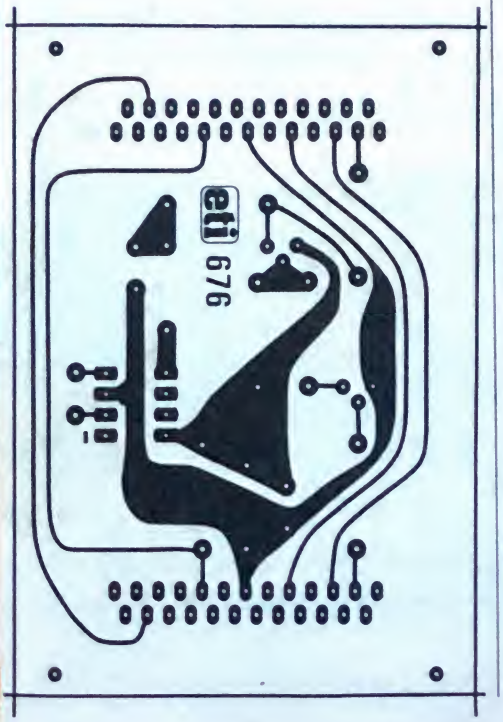
fair dinkum RS232er





Mount the resistors, capacitors and semiconductors first, taking care to correctly orientate the semiconductors and tantalum capacitors. Mount the DB25 plug and socket last — making sure you get each at the correct end of the board. The plug goes at the Microbee end, the socket at the output end. When mounting these, bolt them to the board before soldering the pins so that no stress is placed on the soldered joints.

After a careful check, you're ready to go.



LLIST solver

So, if you are having trouble driving that fancy high quality printer with your 'Bee then check the actual line input printers available that are quite happy to work with only positive-going signals, but occasionally there will be a finicky one that insists on true RS232 levels — in that case, bring out your "Fair Dinkum RS232er" ■

HOW IT WORKS — ETI 676

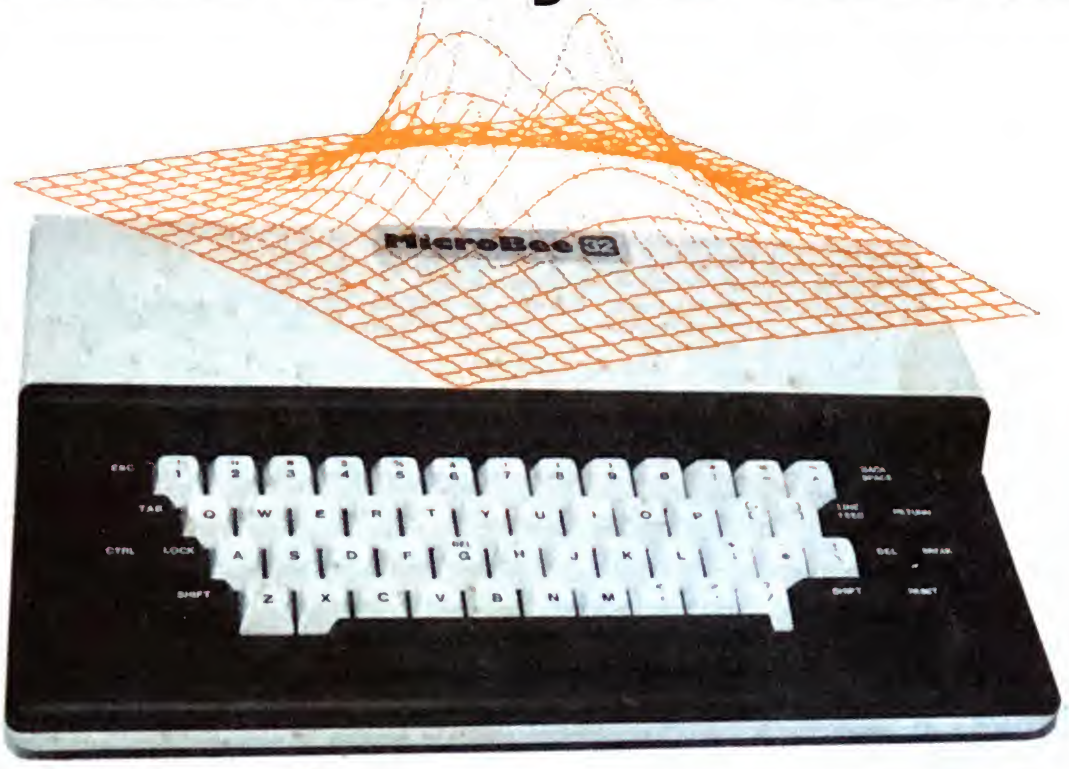
The recipe is simple: take one positive supply rail (from Microbee), regulate it then invert it to provide a negative rail, too. Take common-or-garden NPN/PNP transistor pair and switch TxD signal between positive and negative rails without inversion, Voila — true RS232!

A regulated positive supply rail is supplied by zener ZD1 from the Microbee's internal supply rail (which is around 10 V with a bit of ripple onit). Capacitor C1 provides bypassing.

The negative supply rail is developed by ICl, an Intersil ICL7660 CMOS switching inverter which transfers charge from the positive rail to C2 then in opposite polarity to C3 via D1. The diode is included on the manufacturer's advice to prevent possible destructive latchup of ICl.

The incoming TxD signal from the Microbee is first inverted by Qi, the collector current of which drives the base of Q2 via R4. The emitter of Q2 goes to the unit's positive supply rail, while its collector load (R5) goes to the unit's negative supply rail and thus the TxD signal at the collector of Q2 swings both positive and negative. Resistor R5 provides a measure of protection to the circuit should the output be inadvertently short-circuited.

Plotting three-dimensional surfaces with your Microbee



With the advent of high-resolution graphics on computers, the construction of complex graphs and surfaces is now possible. The Microbee supports 512 x 256 graphics resolution thus giving the owner good scope for the construction of pictures that were not possible a few years ago. The following programs deal with one method to 'plot' three-dimensional surfaces on the Microbee screen and also to a printer.

Jon McCormack

TWO-DIMENSIONAL graphs are a natural for computers since most use a cartesian plane for the 'setting' (turning on) or 'resetting' (turning off) of individual pixels, the dots on the screen. However, when one wants to draw real three-dimensional objects the task becomes more difficult. No matter what method is used, all rely on the use of false perspective — the same as you use to draw 3D pictures on paper. The best method involves feeding in a three-dimensional description of the object to be drawn to the computer. After receiving this information, the computer can then draw a skeletal image of the object. This skeleton is then 'filled in' using complex techniques of shading. (The darker the area the further away it appears.) This method results in a superb illusion of reality. Examples of this can be seen in the Walt Disney film "TRON".

The one disadvantage of this method is that it requires *huge* amounts of memory and very high resolution graphics, both of which are not yet available to the average enthusiast!

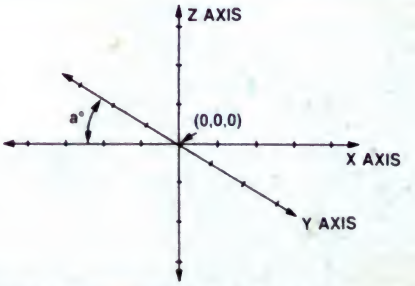


Figure 1. Representation of (X, Y, Z) axis on a 2D plane. The angle 'a' is usually set to 30°.

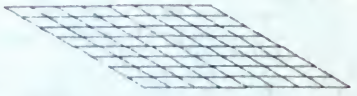


Figure 2. This shows the representation of the surface Z=0, −100<X<100, −100<Y<100. The surface is made up of square 'patches'.

The method used in the following programs just draws the skeletal image and even then does not account for all aspects of perspective drawing. Basically, the program fits a number of square 'patches' to the surface described by a mathemetical equation. The illusion of a third dimension is achieved by placing the y axis at an angle to the x and z axes.

This is shown in Figure 1. The angle 'a' is usually set at 30 degrees. Note that the 'vanishing point' (Where all lines of perspective meet) is assumed to be infinitely far away and thus parallel lines going down the y axis are assumed to never meet. All this is best illustrated by Figure 2 which shows a graph of $z = 0$, where $-100 < x < 100$ and $-100 < y < 100$. The 'patches' can be easily seen. ▶

3D Microbee

3D using hires graphics

The first listing is a program to plot 3D surfaces using standard Microbee HIRES graphics. The equation of the surface to be plotted is put in at line 1020. The most important part of the program is the subroutine at line 2000. This subroutine converts real three-dimensional co-ordinates (x, y, z) to their perspective-corrected equivalents (u, v). The conversion relies on two simple formulae:

$$u = x - (y * \cos(a))$$
and
$$v = z + (y * \sin(a))$$

where 'a' is the angle to the horizontal of the y axis.

The angle 'a' is variable A1 in the program; this is set at line 215. The value in the listing is 30 degrees (don't forget if you wish to change the angle the value is in radians).

The two nested loops in lines 220-270 draw the horizontal lines along the x-axis, the loops in lines 280-340 draw the lines along the y-axis. Operation of the rest of the program can be seen from the comments.

Having typed in the program you are ready to go. First, run the program with the equation shown at line 1020. The program will first ask four questions. For the first two, answer eight. This means there will be eight lines describing the surface plus one end line. For the next two inputs use a domain of −80 to 80 for both. Now sit back and relax and watch your little 'Bee do its stuff. A few moments later you should have the completed surface on your screen.

Try changing the angle of perspective and the equation itself. Note how equations must be 'corrected' — the length of each dot on the screen is equal to one unit. Thus if you wanted to see sin(x) over a complete period, with a domain of −50 to 50 you would have to use sin(x/31.8309) [100/31.8309 = pi, domain is 100]. Some interesting sample equations are listed later on.

Still with the original equation, try an x domain of −200 to 200 and a y domain of −100,100. All goes well until . . . BEEP! You have a graphics error. After the error, if you type "PRINT USED" you will see that all 128 Programmable Graphics Characters (PGC) have been used up. This is the price one must pay for such high resolution. (If the 'Bee still had 512 x 256 resolution and direct-mapped screen memory, 16K of video RAM would be required!).

Wouldn't it be nice to be able to get full 512 x 256 resolution without running out of memory? You can . . . on your printer!

Plot 3D surfaces on your printer

Most modern dot matrix printers are equipped with a feature known as BIT IMAGE GRAPHICS (BIG). This is where the printer stops printing alpha-numeric characters and, instead, will print individual dots according to the input given. Almost all dot matrix printers have this feature

3D surface plotter for standard Microbee hi-res graphics.

```
00100 rem  >>> 3 DIMENSIONAL SURFACE PLOTTER FOR MICROBEE <<<
00110 rem  >> by Jon McCormack, 9 Robinson St., EAST BRIGHTON <<
00120 rem
00130 cls
00140 input "How many lines resolution , x-axis ? ";X
00150 input "How many lines resolution , y-axis ? ";Y
00160 input "What is the domain of x (min,max) ? ";XB,XE
00170 input "What is the domain of y (min,max) ? ";YB,YE
00180 cls : hires
00190 rem  ## Work out values for line positions ##
00200 XP = (XE-XB) / 41(IN) : YP = (YE-YB) / 41(IN)
00210 A = 256 : B = 128
00215 A1 = 0.5235987 : S1 = sin(A1) : C1 = cos(A1) : rem ## A1 is perspective
  angle to horizontal ##
00220 for Y2 = YB to YE step YP
00230 for X2 = XB to XE
00240 gosub (X2,Y2) 1000 : rem ## Work out equation ##
00250 gosub (X2,Y2,Z2) 2000 : rem ## Convert 3D co-ordinates to 2D equivalents ##
00260 set A + U , B + V / 2
00270 next X2 : next Y2
00280 rem ## Now plot lines along y axis ##
00290 for X2 = XB to XE step XP
00300 for Y2 = YB to YE
00310 gosub (X2,Y2) 1000 : rem ## Work out equation for these values of xb,y2 ##
00320 gosub (X2,Y2,Z2) 2000 : rem ## Convert 3D co-ordinates to 2D equivalents ##
00330 set A + U , B + V / 2
00340 next Y2 : next X2
00350 stop
01000 rem ## This subroutine has the equation to be calculated ##
01010 var (X2,Y2)
01020 Z2 = 150*((X2*X2/1000)+(Y2*Y2/500))
01030 return
02000 rem ## This subroutine converts 3D co-ordinates to perspective drawn 2D
  plotting positions ##
02010 rem ## Variable A1 is the value of the slope ##
02020 var (U2,V2,Z2)
02030 U2 = X2 - Y2 * C1 : rem ## C1 is cos(A1) ##
02040 V2 = Z2 + Y2 * S1 : rem ## S1 is sin(A1) ##
02050 U = int(U2) : V = int(V2)
02060 return
```

Listing 2a. BASIC program to print 3D surfaces to your printer.

```
00100 REM  >>> 3 DIMENSIONAL SURFACE PRINTER FOR MICROBEE <<<
00110 REM  >> by Jon McCormack, 9 Robinson St., EAST BRIGHTON <<
00120 REM
00130 CLS
00140 INPUT "How many lines resolution , x-axis ? ";X
00150 INPUT "How many lines resolution , y-axis ? ";Y
00160 INPUT "What is the domain of x (min,max) ? ";XB,XE
00170 INPUT "What is the domain of y (min,max) ? ";YB,YE
00190 REM  ## Work out values for line positions ##
00200 XP = (XE-XB) / 41.7(IN) : YP = (YE-YB) / 41.7(IN)
00210 A = 512 : B = 256 : G = 8192 : F = B / 8 % A : GOSUB 3000
00212 C = A / 2 : D = B / 2 : REM ## position for (0,0,0) ##
00215 A1 = 0.5235987 : S1 = SIN(A1) : C1 = COS(A1) : REM ## A1 is perspective
  angle to horizontal ##
00220 FOR Y2 = YB TO YE STEP YP
00230 FOR X2 = XB TO XE
00240 GOSUB (X2,Y2) 1000 : REM ## Work out equation ##
00250 GOSUB (X2,Y2,Z2) 2000 : REM ## Convert 3D co-ordinates to 2D equivalents ##
00260 GOSUB (C+U,D+V) 4000
00270 NEXT X2 : NEXT Y2
00280 REM ## Now plot lines along y axis ##
00290 FOR X2 = XB TO XE STEP XP
00300 FOR Y2 = YB TO YE
00310 GOSUB (X2,Y2) 1000 : REM ## Work out equation for these values of xb,y2 ##
00320 GOSUB (X2,Y2,Z2) 2000 : REM ## Convert 3D co-ordinates to 2D equivalents ##
00330 GOSUB (C+U,D+V) 4000
00340 NEXT Y2 : NEXT X2
00345 GOSUB 5000
00350 STOP
01000 REM ## This subroutine has the equation to be calculated ##
01010 CMR (X2,Y2)
01020 Z2=1:REM Z2 = 100/((X2*X2/1000)+(Y2*Y2/500)+1)
01030 RETURN
02000 REM ## This subroutine converts 3D co-ordinates to perspective drawn 2D
  plotting positions ##
02010 REM ## Variable A1 is the value of the slope ##
02020 VAR (X2,Y2,Z2)
02030 U2 = X2 - Y2 * C1 : REM ## C1 is COS(A1) ##
02040 V2 = Z2 + Y2 * S1 : REM ## S1 is SIN(A1) ##
02050 U = INT(U2) : V = INT(V2)
02060 RETURN
```

continued . . .

BASIC program to print 3D surfaces to your printer.

```
03000 REM **  CLEAR MEMORY VIA MACHINE LANGUAGE SUBROUTINE  **
03005 PRINT "Clearing memory , please wait."
03010 USR(7680,F)
03020 DIM W(8)
03030 FOR X = 0 TO 7
03040 READ W(X) : NEXT X
03050 DATA 1,2,4,8,16,32,64,128
03060 PRINT "Ready." : RETURN
04000 REM ** This subroutine stores a given value into memory **
04010 VAR (0,P)
04015 CURS 20,8:PRINT "co-ordinates : ";Q;" , ";P;"     "
04017 IF Q<0 OR Q>A OR P<0 OR P>B THEN RETURN
04020 Z = P / 8
04030 I =  P - Z * 8
04040 G = A * Z + A - Q
04050 H = PEEK(E+F-G)
04055 K = -(H AND W(I))
04060 IF K THEN RETURN : REM  * If bit set then return *
04070 H = H + W(I) : POKE  E+F-G,H
04080 RETURN
05000 REM ** This subroutine prints the graph to the printer **
05010 OUTL#1 : REM ** Set printer output **
05020 LPRINT "3D Surface plotter by Jon McCormack."
05025 POKE 7768,B/8 : USR(7692,A)
05030 LPRINT
05035 LPRINT CHR$(27);"@";
05040 RETURN
```

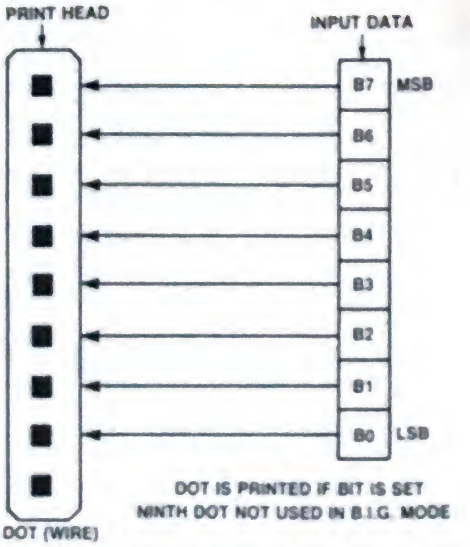


Figure 3. Diagram of the print head. Once set into the Bit Image Graphics mode ('ESC' 'K' is sent to the printer to set up this mode), the individual bits in each byte of input data control the setting of the dots on the printer.

although the method of getting into this mode and printing dots differs on different printers. Bit image graphics are not possible on daisy wheel printers.

The second program shows how to draw 3D surfaces on your printer. For this program to operate correctly you will need the following:

1. A 32K Microbee. It is not essential to have 32K but in order to get full 512 x 256 resolution, 32K is required. With 16K only 256 x 256 or 512 x 128 resolution can be obtained.

2. An Admate DP-80 dot matrix printer. (This is the printer sold by Applied Technology for the Microbee.) However if you have another printer don't go out and sell it yet. the DP-80 is identical to almost all the XYZ-80 printers (this includes FAX-80, BX-80, ALPHA-80, CP-80 etc., etc.) and I understand (though I haven't tested it) even the Epson MX-80. Even if you don't have any of these printers the program can be modified to accept other printers.

3. Time. Depending on the complexity of the equation the computer may take up to half an hour to come up with the goods. It is a good idea to have the kettle and a pack of cards at the ready.

Now onto the program — it is in two parts. The first is the BASIC listing which does most of the maths. The second is in machine language and performs the things that the Z-80 does best: clearing and moving memory. If you can't be bothered typing in all the listings, send $9 to the author for a cassette of all the programs listed. (Don't forget to tell me your address!)

The BASIC part appears similar to the first listing for screen graphics — essentially the same method is used. However, instead of setting graphics dots on the screen the dot is set in the Bee's memory. To see how this works it is necessary to understand how the printer prints bit image graphics.

Figure 3 shows a diagram of the print head. The head has nine wires which print the dots on the paper. Normally, the head prints a series of seven columns of these dots to form a letter or number. The sequence of printing these columns of dots is controlled by the printer's internal microprocessor. In the bit image mode the user (computer) controls the sequence in which the dots are set. According to the value sent to the printer while in the BIG mode certain dots are set. In this mode only eight of the nine dots are used. Because of this it is necessary to change the length of a form feed entering the BIG mode. The Admate printer has two different graphics modes; one with 640 x 8 dots per line (lores) and 1280 x 8 dots per line (hires). For most applications the lores mode is enough.

Before the 3D 'picture' can be printed the whole thing is stored in a huge chunk of memory (16K for 512 x 256) by a method known as *bit mapping*. Each bit represents a graphics pixel. If the pixel is on then the bit is set. Now there are eight bits in one byte — the same number as there are wires in the print head. Thus, each byte prints one eight-dot column and 512 of these make up one line. Since each byte in the memory array used to store the graph before printing is contiguous, dumping the memory in 512 byte blocks will print the 3D surface. If you are in the dark about how this all works then take a look at Figure 4.

How the program works

As mentioned earlier the basic part of the program is very similar to the first program. Instead of setting the point calculated on the video screen it is set in the 512 x 256 bit grid of Microbee memory. The subroutine at line 4000 does this. The rest of the BASIC part of the program can be understood from the comments and the first listing. The Machine code listing contains two main subroutines that are called by the BASIC program. The first, CLEAR, clears the block of memory to be used to store the surface before printing. The amount of memory to be cleared is sent to the BC registers via the BASIC 'USR' command (line 3010). The amount needed is calculated from the formula m = (x * y)/8. Where x is the number of dots available on the x-axis and y is the dots on the y-axis. ▶

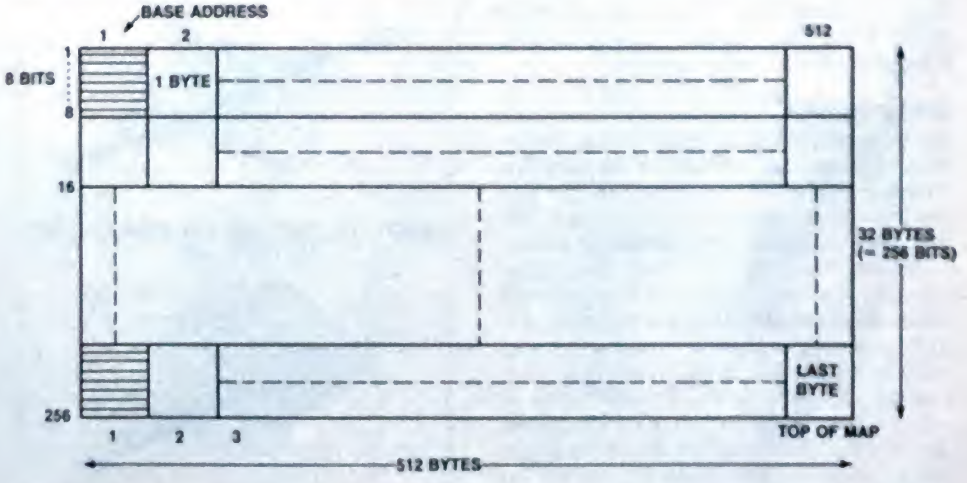


Figure 4. This diagram shows how Microbee memory is used as a bit map. Each bit represents one graphics pixel. In the diagram the size shown is 512 x 256, however, this can be changed (see text).

3D Microbee

The second machine code subroutine dumps the finished memory contents to the printer. The subroutine requires two values. The first, the number of columns, is sent to the BC registers via the USR command. The second, the number of 8-bit rows, is POKEd into a memory location called ROWS (at 1E58 hex). Once given these values the routine will print out the graph.

The plus signs ('+') seen on the output are for alignment — they show the edge of the field. The other subroutines in the machine code listing are internal and are called by DUMP. To enter in the program, first type in the assembly listing (either by EDASM or the MONITOR), then the BASIC listing. It is unwise to use DATA statements to poke in the machine code part under BASIC as this uses too much memory.

How to change the parameters

There are several variables that can be changed to suit your particular amount of memory. Line 210 in the BASIC listing contains the important variables. A is the 2D x-axis length — in this case it's 512 (same as the Bee's graphics). B is the y-axis length. E is the BASE address for the start of storage memory. It is set at 2000 hex (8192). However, if you have a 16K Microbee you may have to lower this value, but check that it does not overlap with the BASIC program and its storage required for variables. Also, if you change the value of E then make sure you also change the value of BASE in the machine code listing to the same value. The value of F is the total amount of memory required.

Changing the value of A and B at line 210 is accounted for in the rest of the program. Make sure that B is a multiple of 8 (eg. 64, 128, 256 etc.).

Well, that's about it. Try changing the values — many different results can be obtained. Note that, for complex equations some time is required for the computer to work out all the different values (sometimes up to 30-40 minutes!!) so when testing equations either do so on the screen version or enter a small domain and low number of lines resolution to get a vague idea what the graph looks like. Figure 5 shows various examples of the results obtained.

Extensions

To get a really high number of dots resolution requires huge amounts of memory (for example 1024 x 1024 requires 128K!!!). You can still get this resolution even with a 16K Microbee by printing the surface in parts.

Delete lines 5020, 5030 and 5035. Now, suppose you want 512 x 512 but you only have enough memory to print 512 x 256. All you have to do is run the program as normal, assuming 512 x 512 resolution, only change line 4017 in the BASIC listing to reject values in the bottom 512 x 256 part of the bit map. Now the top part of the graph will be printed. Again, changing line 4017 to reject the values in the upper 512 x 256 block will then append the bottom part to the top. ▶

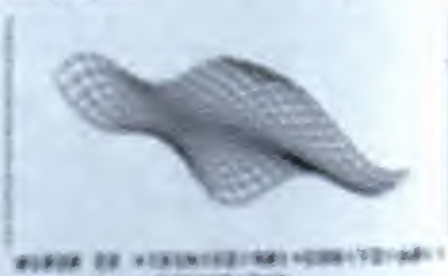
Listing 2b. Machine code listing.

The subroutines are called by the BASIC program, Listing 2a. Listing 2b should be assembled first, then Listing 2a typed in.

```
ADDR   CODE   LINE   LABEL  MNEM   OPERANDS
       ; 3 DIMENSIONAL SURFACE PLOTTER BY JON [illegible] .
       ; [illegible]
       ;
       ; MACHINE LANGUAGE SUBROUTINES (CALLED FROM BASIC).
       ;
       ; SET UP VALUES FOR ASSEMBLER ;
       ;
BASE    EQU    [illegible]      ;BASE ADDRESS FOR STORAGE
ESC     EQU    27
PRINT   EQU    [illegible]      ;BASIC ROM ROUTINE
       ;
       ;
       ;
       ;
       ;
       ;        CLEAR SUBROUTINE
       ; This subroutine clears memory from address BASE to value
       ; in the BC registers.
       ;        To use : CALL CLEAR
       ;
        ORG    [illegible]
CLEAR   LD     HL,BASE
NEXT    LD     (HL),0           ;ZERO BYTE IN MEMORY
        INC    HL               ;MOVE TO NEXT POSITION
        DEC    BC
        LD     A,B
        OR     C
        JR     NZ,NEXT          ;CLEAR UNTIL FINISHED
        RET                     ;END CLEAR
       ;
       ;
       ;        DUMP SUBROUTINE
       ; This subroutine dumps memory to the printer. No. of rows
       ; is specified by memory location ROWS and columns by BC.
       ;        Usage : CALL DUMP
       ;
DUMP    LD     A,ESC            ;SET UP
        CALL   PRINT            ;PRINTER FOR
        LD     A,[illegible]    ;BIT IMAGE GRAPHICS
        CALL   PRINT
        PUSH   BC
        CALL   [illegible]
        LD     HL,ROWS
        LD     B,(HL)           ;PUT NO. OF ROWS IN B
        LD     HL,BASE
LOOP    CALL   SETUP
        POP    DE               ;NO. OF COLUMNS TO PRINT
        PUSH   DE               ;SAVE FOR NEXT TIME ROUND
CLOOP   LD     A,(HL)           ;PRINT BYTE
        CALL   PRINT
        INC    HL
```

Figure 5. The 15 plots here and four over the page show the 'surfaces' produced by a variety of equations, illustrating the capabilities of the program.

[illegible]

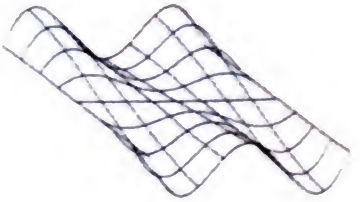


[illegible]

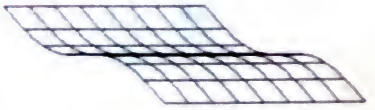
[illegible]

[illegible]

```
1E2B 1B         00810           DEC     DE
1E2C 7B         00820           LD      A,E
1E2D B2         00830           OR      D
1E2E 20F6       00840           JR      NZ,OLOOP        ;IF LINE NOT FINISHED CONTIN
UE
1E30 3E2B       00845           LD      A,'+'
1E32 CD4580     00847           CALL    PRINT
1E35 10EA       00850           DJNZ    LOOP            ;PRINT NEXT LINE UNTIL DONE
1E37 D1         00860           POP     DE              ;CORRECT STACK
1E38 C9         00870           RET                     ;END DUMP
                00880 ;
                00890 ;         SETUP SUBROUTINE
                00900 ; Sets up printer for Bit Image Graphics. Called by DUMP.
                00910 ;
1E39 E5         00915 SETUP     PUSH    HL              ;SAVE HL AND BC
1E3A C5         00917           PUSH    BC
1E3B 0607       00920           LD      B,7
1E3D 21511E     00930           LD      HL,SETTBL
1E40 7E         00940 NEXTCH    LD      A,(HL)          ;SET UP PRINTER
1E41 CD4580     00950           CALL    PRINT           ;FOR B.I.G. MODE
1E44 23         00960           INC     HL
1E45 10F9       00970           DJNZ    NEXTCH
1E47 C1         00974           POP     BC
1E48 E1         00976           POP     HL
1E49 C9         00980           RET                     ;END SETUP
                00990 ;
                01000 ;         CALC SUBROUTINE
                01010 ; This subroutine works out values for No. of Bit image
                01020 ; charactors to send to the printer. Called by DUMP.
                01030 ;
1E4A 21561E     01040 CALC      LD      HL,VALUES
1E4D 71         01050           LD      (HL),C          ;STORE VALUES
1E4E 23         01055           INC     HL
1E4F 70         01057           LD      (HL),B
1E50 C9         01060           RET                     ;END CALC
                01070 ;
                01080 ; Printer output table for ADMATE DP-80 Printer.
                01090 ;
1E51 0A0D       01100 SETTBL    DEFW    0D0AH           ;CAR.RET. / LINE FEED
1E53 2B         01110           DEFB    '+'
1E54 1B         01120           DEFB    ESC
1E55 4B         01130           DEFB    'K'             ;SET UP BIT IMAGE MODE
1E56 0000       01140 VALUES    DEFW    0000H
                01150 ;
                01160 ; TEMP MEMORY LOCATION USED BY DUMP
                01170 ;
1E58 1F         01190 ROWS      DEFB    1FH
                01200 ;
                01210 ;
001F            01220           END                     ;END SUBROUTINES.
00000 Total errors

VALUES  1E56    NEXTCH  1E40    SETTBL  1E51    OLOOP   1E26
SETUP   1E39    LOOP    1E21    ROWS    1E58    CALC    1E4A
DUMP    1E0C    NEXT    1E03    CLEAR   1E00    PRINT   8045
ESC     001B    BASE    2000
```

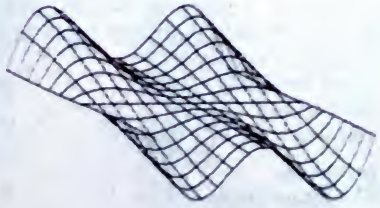


Plot of the equation in the listing.

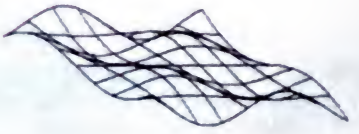


Perspective angle at 10°.

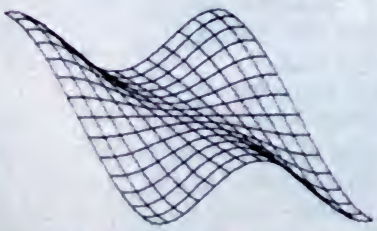
Perspective angle at 20°.



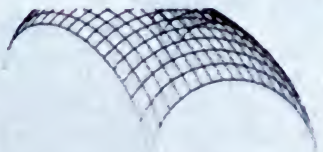
```
01020 Z2 =SQR(-X2*X2-Y2*Y2+5000)
```



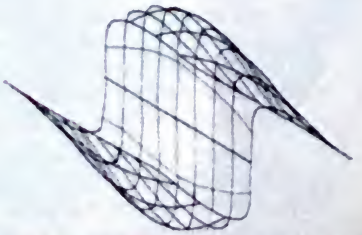
Perspective angle at 30°.



```
01020 Z2 =SIN(X2/60)*COS(Y2/110)
      *Y2*X2/100
```



```
01020 Z2 =SQR(-X2*X2-Y2*Y2+20000)
```



Dots in the centre area were so far apart I joined them by pen.

3D Microbee

Dots at the centre joined by pen.

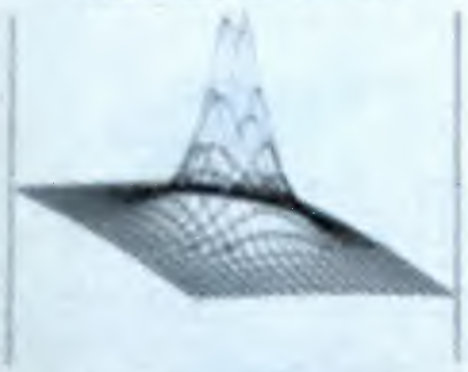


Figure 8. Full 512 x 512 dot resolution. This was obtained by printing 512 x 256 'chunks' of the complete surface (see text).

This process is illustrated in Figure 8. There is no reason why you can't break up a large resolution into more than two blocks, except that it will take a lot of time to produce very high resolution surfaces. However, this time is well worth the effort as can be seen from some of the results shown. ■

A tape listing of the programs here can be obtained for $9.00 from the author at 9 Robinson St, East Brighton, Vic. 3187.

SHOPAROUND

This page is to assist readers in the continual search for components, kits, printed circuit boards and other parts for ETI projects and circuits.

ETI-649 light pen

This straightforward project should present few problems to constructors. Most parts are commonly available, though you might have to shop around for the phototransistor. The LPT100 should be stocked by *Dick Smith stores*, *Altronics* (Perth), *Jaycar* (Sydney), *Rod Irving Electronics* (Melbourne), *Elliotronics* (Melbourne) and *Sheridan Electronics* (Sydney). *Promark Electronics* in Sydney and Melbourne should be able to supply the BPX25 or a suitable equivalent.

The case shown in the project was obtained from *Jaycar* but is also available through *Altronics*.

Kits are stocked by *Altronics* in Perth, *Jaycar* in Sydney and *Rod Irving Electronics* in Melbourne.

ETI-671 Parallel printer interface

All the parts are widely obtainable so constructors should experience no difficulty. Kits should be available from *Altronics* in Perth, *Jaycar* in Sydney, *Rod Irving Electronics* and Magraths in Melbourne.

The 40-column Multitech printer shown in the article is obtainable from *Emona Enterprises* in Sydney, *Suite 204/661 George St, Sydney 2000 NSW. (02)212-4815.*

ETI-672 TTY interface

The cheap way to get hard copy is to attach an old teletype. This project is available in kit form from the suppliers listed above. Components are widely available.

ETI-673 MultiPROM board

This project, from Aztek, is distributed by *Jaycar* in Sydney. The pc board copyright has been retained by the designer and artwork is not available.

ETI-674 joystick interface

Kits for this project are produced by *Altronics* in Perth, and *Rod Irving Electronics* in Melbourne. Try *Magraths* in Melbourne, also.

ETI-675 Serial/parallel interface

The GI AY-3-1015D UART is distributed by *Dataroa* in Melbourne. Kits should be obtainable from *Altronics* in Perth and *Rod Irving Electronics* in Melbourne.

ETI-676 RS232er

For kits of this project, try *Magraths* and *Rod Irving Electronics* in Melbourne. In Sydney, try *Jaycar* for the bits. The Intersil ICL7660 chip is imported and distributed by *RAD Electronics* in Melbourne. *All Electronic Components* in Melbourne and *Geoff Wood Electronics* in Sydney are their retail distributors.

ETI-678 ROM reader

Parts for this should be readily available. Kits can be obtained from *Altronics* in Perth, *Jaycar* in Sydney and *Rod Irving Electronics* in Melbourne. Try *All Electronic Components*, too.

ETI-733 RTTY decoder

A well-supported project! A you-beaut kit for this, housed in an all-plastic jiffy box with silk-screened front panel is produced by *Altronics* in Perth. Other kit suppliers include *Rod Irving Electronics* in Melbourne, and *Jaycar* in Sydney. You might also try *Magraths* and *All Electronic Components* in Melbourne.

ETI-736 FAX decoder

Be your own weather forecaster? Kit suppliers for this are the same as listed above.

Artwork

Artwork for the pc boards and/or front panels for these projects is available from the publishers. When ordering, ensure you quote the project number(s) and name(s) and state whether you want positive or negative film transparencies, according to what your process requires. Prices (post paid) are as follows:

ETI-649	$2.00
ETI-671	$1.00
ETI-672	$1.00
ETI-674	$2.70
ETI-675	$4.00
ETI-676	$2.00
ETI-678	$1.60
ETI-733	$3.60
ETI-736	$1.50

Orders should be sent to:

ETI Artwork Sales
PO Box 227
Waterloo NSW 2017

Make your cheque or money order payable to "ETI Artwork Sales".

Printed Circuit boards

Ready-made pc boards can be obtained from the following suppliers; apart from the kit suppliers mentioned.

Jemal Products
PO Box 168
Victoria Park WA 6100

Jaetronics
58 Appian Drive
St Albans Vic 3021

Acetronics
112 Robertson Rd
Bass Hill NSW 2197

RCS Radio
651 Forest Rd
Bexley NSW 2207

Mini Tech
PO Box 9194
Auckland N.Z. ■

Machine Language Locations

The following listings are reprinted, with the kind permission of Applied Technology, from the 'Microbee Technical Manual'. We hope they'll save you many of the tedious parts of programming and let you get on with something really creative.

0	0000	HIRES/Cassette scratch; buffer for IN/OUT#
128	0080	Interrupt vectors
136	0088	PIO Interrupt vectors
138	008A	Network interrupt vector LSB+1
140	008C	Break key: 0=enable, 1=disable
141	008D	0 to enable EDIT/SAVE/LIST; Non zero to disable
160	00A0	Top of RAM pointer LSB+1
162	00A2	Warm start jump address LSB+1
164	00A4	Initial check byte LSB+1: 55A for warm start
166	00A6	Machine Language EXEC address
168	00A8	Jump vector for MEM/PAK
171	00AB	Jump vector for NET
174	00AE	Jump vector for EDASM
178	00B2	VDU output vector LSB+1
180	00B4	Parallel port output vector LSB+1
182	00B6	300 baud Cassette output vector LSB+1
184	00B8	1200 baud Cassette output vector LSB+1
186	00BA	300 baud RS232 output vector LSB+1
188	00BC	1200 baud RS232 output vector LSB+1
190	00BE	Reserved output vector LSB+1
192	00C0	Reserved output vector LSB+1
194	00C2	Keyboard input vector LSB+1
196	00C4	Parallel port input vector LSB+1
198	00C6	300 Cassette input vector LSB+1
200	00C8	1200 baud Cassette input vector LSB+1
202	00CA	300 baud RS232 input vector LSB+1
204	00CC	1200 baud RS232 input vector LSB+1
206	00CE	Reserved input vector LSB+1
208	00D0	Reserved input vector LSB+1
210	00D2	6545 R0 Total # of Horiz chars -1 per line; Determines horizontal sync frequency
211	00D3	6545 R1 Total # of displayed chars per line
212	00D4	6545 R2 Pos of horiz sync pulse on a line
213	00D5	6545 R3 Vert and horiz sync widths (2 nibbles)
214	00D6	6545 R4 Vert total rows per screen -1
215	00D7	6545 R5 Vert total line adjust
216	00D8	6545 R6 Total # of vert displayed rows
217	00D9	6545 R7 Vert sync position
218	00DA	6545 R8 Mode control register
219	00DB	6545 R9 # of scan lines per character

▶

Machine Language Locations

DEC	HEX	DESCRIPTION
220	ØØDC	6545 R1Ø Cursor start line and blink mode
221	ØØDD	6545 R11 Cursor end line
222	ØØDE	6545 R12 Display start address (high)
223	ØØDF	6545 R13 Display start address (low)
224	ØØEØ	6545 R14 Cursor position (high)
225	ØØE1	6545 R15 Cursor position (low)
226	ØØE2	Output device byte
227	ØØE3	List output device byte
228	ØØE4	Input device byte
229	ØØE5	Video mode byte
230	ØØE6	Output speed byte
231	ØØE7	PCG characters used
232	ØØE8	PLOT type (H=48,I=49,R=52)
233	ØØE9	Tape baud: 1=1200 4=300
234	ØØEA	RS232 baud: 1=1200 Ø=300
235	ØØEB	PLOT/Tape buffer (till Ø1ØØ):
256	Ø1ØØ	Start of CP/M Transient Program area (till BFFF)
256	Ø1ØØ	PLOT/Tape buffer (from ØØEB):
		PLOT: ØØEB - ØØF8 Scratch area
		ØØF9 - ØØFA end X
		ØØFB - ØØFC end Y
		ØØFD - ØØFE start X
		ØØFF - Ø1ØØ start Y
		Tape: ØØEB - ØØFØ Name of file to load
		ØØF1 - ØØF6 Name of file loaded
		ØØF7 File type
		ØØF8 - ØØF9 Length of file
		ØØFA - ØØFB Start address
		ØØFC - ØØFD Auto-start address
		ØØFE Speed (FF=1200,ØØ=300 bd)
		ØØFF Auto-start flag
		Ø1ØØ (spare)
257	Ø1Ø1	Shift Lock Ø=Lower case, 1=Upper case
258	Ø1Ø2	Keyboard auto-repeat delay, 255=off, 52=on
259	Ø1Ø3	Delay for keyboard debounce LSB+1
262	Ø1Ø6	ASCII value of last key pressed
263	Ø1Ø7	(as above but used by KEY$)
264	Ø1Ø8	External keyboard buffer
265	Ø1Ø9	Handshake byte for par port out; zero if usable
266	Ø1ØA	Number of bytes in Cassette redirect buffer
267	Ø1ØB	CURS position buffer LSB+1
269	Ø1ØD	NZ if last VDU out char was esc
270	Ø1ØE	Save text pointer in graphics
273	Ø111	List buffer size: Initialised as B8 by NEW
274	Ø112	Save BASIC text pointer
278	Ø116	Scratch for RENUM
279	Ø117	Scratch for RENUM
280	Ø118	Save start of REAL number
282	Ø11A	Random number seed (till Ø129)
297	Ø129	Random number seed (till Ø11A)

DEC	HEX	DESCRIPTION
298	Ø12A	LOAD type: Ø=LOAD or SAVE, 1=LOAD?, 2=LOADU
318	Ø13E	Save text pointer
319	Ø13F	CONTinue address LSB+1
320	Ø140	Address of current line LSB+1
322	Ø142	Line number if ON ERROR active LSB+1
324	Ø144	Which error message - ERRORC
325	Ø145	ERRORL LSB+1
512	Ø200	HIRES/PCG scratch (till Ø3FF)
512	Ø200	Length of Source file
561	Ø231	Pointer to end of Source file
603	Ø25B	Monitor Scratch pad 6
605	Ø25D	Monitor Scratch pad 7
768	Ø300	Cassette buffer
784	Ø314	EDASM Monitor Tape Head buffer
932	Ø3A4	Top of Monitor stack
1023	Ø3FF	HIRES/PCG scatch (from Ø200)
1024	Ø400	FOR...NEXT stack for BASIC (till Ø4FF)
1279	Ø4FF	FOR...NEXT stack for BASIC (from Ø400)
1280	Ø500	Start of 208 2 byte pointers for Real variables
1280	Ø500	Word-bee file initialised flag
1299	Ø513	Word-bee top of memory pointer LSB+1
1303	Ø517	Word-bee start of file pointer LSB+1
1309	Ø51D	Word-bee end of file pointer LSB+1
1335	Ø537	Word-bee Monitor Tape Header buffer
1392	Ø570	Location of cursor in Monitor mode LSB+1
1395	Ø573	Address of Monitor initial entry e.g. E or A command
1712	Ø6BØ	Start of 26 2 byte pointers for Interger variables
1764	Ø6E4	Address of line containing FNØ LSB+1
1766	Ø6E4	Address of line containing FN1 LSB+1
1768	Ø6E8	Address of line containing FN2 LSB+1
1770	Ø6EA	Address of line containing FN3 LSB+1
1772	Ø6EC	Address of line containing FN4 LSB+1
1774	Ø6EE	Address of line containing FN5 LSB+1
1776	Ø6F1	Address of line containing FN6 LSB+1
1778	Ø6F3	Address of line containing FN7 LSB+1
1832	Ø728	208 char Input/Program line buffer (till Ø7EØ)
2016	Ø7EØ	208 char Input/Program line buffer (from Ø728)
2048	Ø8ØØ	192 character list output buffer
2240	Ø8CØ	No. of sig. digits (must be < 62) - Start prog info
2243	Ø8C3	Input edit buffer pointer
2245	Ø8C5	Input buffer pointer LSB+1
2247	Ø8C7	Line size
2250	Ø8CA	Current line
2254	Ø8CE	Top of string pointer LSB+1
2256	Ø8DØ	Program Begin pointer LSB+1
2258	Ø8D2	Program End pointer LSB+1 (for recovery)
2260	Ø8D4	Top of for...Next stack LSB+1
2262	Ø8D6	Output pointer
2264	Ø8D8	Next available location for storing variable LSB+1
2268	Ø8DC	String pointer: top of free memory
2272	Ø8EØ	PRMT charater
2273	Ø8E1	Systerm prompt character ">"
2274	Ø8E2	Line size
2275	Ø8E3	Zone width

Machine Language Locations

DEC	HEX	DESCRIPTION
2276	08E4	Variable key
2281	08E9	Data pointer LSB+1
2284	08EC	Lower limit of string area LSB+1
2286	08EE	AUTO increment
2287	08EF	AUTO current line number LSB+1
2293	08F5	POS LSB+1
2295	08F7	Default line for EDIT LSB+1
2299	08FB	No of GOSUBs active
2300	08FC	0=LPRINT, 1=PRINT, bit 7 set for TRACE
2301	08FD	CONT line number LSB+1
2303	08FF	(unused)
2304	0900	Start of BASIC program Start of Word-bee file (must be 00 hex)
2305	0901	First character in Word-bee file
8191	1FFF	End of RAM in 8k machine
16128	3F00	Stack/top of strings for 16k
16383	3FFF	End of RAM in 16k machine
32512	7F00	Stack/top of strings for 32k
32767	7FFF	End of RAM in 32k machine
32768	8000	Start of BASIC
32771	8003	BASIC warm start
32774	8006	Wait for Keyboard input - A register
32777	8009	Scan Keyboard
32780	800C	Display character in B register
32783	800F	(unused)
32786	8012	Get byte from cassette
32789	8015	Get B bytes from cassette
32792	8018	Cassette byte out A
32795	801B	B bytes out to cassette from (HL)
32798	801E	Auto-execute address for saving BASIC program
32801	8021	Warm start for restoring Reset jump
32804	8024	HIRES initialisation
32807	8027	LORES initialisation
32810	802A	INVERSE initialisation
32813	802D	UNDERLINE initialisation
32816	8030	SET dot: X = HL, Y = DE
32819	8033	RESET dot returns Z if OK
32822	8036	INVERT dot
32825	8039	Test for dot - NZ if set/error
32828	803C	PLOT a line
32831	803F	Redirected input A
32834	8042	Redirected output A
32837	8045	Redirected print output A
32804	8048	Jump to BASIC with CLEAR
32843	804B	Jump to BASIC command level
32846	804E	Jump to BASIC after RET or NEW
32849	8051	Check buffer
32852	8054	Insert new line

DEC	HEX	DESCRIPTION
32855	8057	Line into list buffer
32858	805A	Option not fitted error
32861	805D	Syntax error
32908	808C	Print "?"
32912	8090	Print a space
32914	8092	Display a char in A register on printer or VDU
33349	8245	Skip blanks
33751	83D7	Jump vector for RUN
33852	843C	Prompt byte - 3E hex for ">"
33908	8474	Take token in A reg. and return Jump vector in HL
33934	848E	Clear variables
34052	8504	Check keys pressed with <RESET>
34071	8517	Monitor to BASIC warm start
34074	851A	Return to BASIC without corrupting E2 or E4
34091	852B	Clear first 900 hex bytes
34104	8538	Initialise keyboard code
34109	853D	Initialise vectors
34151	8567	Find top of memory
34183	8587	Jump vector for NEW
34215	85A7	Load 16 bytes addressed by HL into 6545 registers
34230	85B6	Jump vector for LLIST
34238	85BE	Jump vector for LIST
36968	9034	Jump vector for LET
36938	904A	Jump vector for LPRINT
36946	9052	Jump vector for PRINT
36985	9479	Jump vector for IF
38044	949C	Jump vector for GOTO
38381	95ED	Jump vector for FOR
38590	96BE	Jump vector for NEXT
38814	979E	Jump vector for INPUT
38950	9826	Jump vector for DATA
39015	9867	Jump vector for READ
39090	98B2	Jump vector for DIM
39289	9979	Jump vector for END
39295	997F	Jump vector for STOP
39340	99AC	Print text pointed to by HL register
39368	99C8	Jump vector for AND, CHR, ELSE, KEY, NOT, OFF OR, SPC, STEP, STR, TAB, THEN and TO
39371	99CB	Jump vector for POKE
39398	99E6	Jump vector for OUT
39512	9A58	Jump vector for ZONE
39530	9A6A	Jump vector for SD
39550	9A7E	Jump vector for RESTORE
39582	9A9E	Jump vector for PRMT
39596	9AAC	Jump vector for CLEAR
39627	9ABC	Jump vector for IN
39662	9AEE	Jump vector for INT
39674	9AFA	Jump vector for PEEK
39683	9B03	Jump vector for USR
39720	9B28	Jump vector for LEN
39753	9B49	Jump vector for ABS
39773	9B5D	Jump vector for RND
39828	9B94	Jump vector for FLT
39861	9BB5	Jump vector for FRE

Machine Language Locations

DEC	HEX	DESCRIPTION
39901	9BDD	Jump vector for SQR
40034	9C62	Jump vector for GOSUB
40105	9CA9	Jump vector for RETURN
40126	9CBE	Jump vector for VAR
40174	9CEE	Jump vector for FRACT
40200	9D08	Jump vector for SGN
40300	9D6C	Jump vector for SEARCH
40360	9DA8	Jump vector for STRS
40387	9DC3	Jump vector for VAL
40562	9E72	Jump vector for SIN
40678	9EE6	Jump vector for ATAN
40659	9ED3	Jump vector for COS
40797	9E5D	Jump vector for LOG
40939	9FEB	Jump vector for EXP
41095	A087	Jump vector for ON
41265	A131	Jump vector for TRACE
41322	A16A	Jump vector for CONT
41366	A196	Jump vector for POS
41381	A1A5	Jump vector for ASC
41399	A1B7	Jump vector for FN
41429	A1D5	Jump vector for REM
41667	A2C3	Jump vector for EDIT
41870	A38E	Print text pointed to by HL register
41941	A3D5	Parrallel port input vector
41961	A3E9	Keyboard input vector
41971	A3F3	Test for key repeat 14 times
42238	A4FE	Test shift key
42244	A504	Test control key
42250	A50A	Test for particular key pressed
42434	A5C2	Line input of text
42523	A61B	Jump vector for CLS
42541	A62D	Call address to print a space
42523	A62F	VDU out - A register
42724	A6E4	Shift screen
42772	A714	Jump vector for PLAY
42847	A75F	Generate one of 255 poss tones
42877	A77D	Jump vector for SPEED
42894	A78E	Set cursor position
42925	A7AD	Turn off cursor
42936	A7B8	Jump vector for NORMAL
42944	A7C0	Jump vector for PCG
42952	A7C8	Jump vector for UNDERLINE
43003	A7FB	Jump vector for INVERSE
43052	A82C	300 baud RS232 input vector
43057	A831	1200 baud RS232 input vector
43130	A87A	300 baud RS232 output vector
43135	A87F	1200 baud RS232 output vector
43212	A8CC	Get program name
43234	A8E2	Compare program names
43260	A8FC	Display '*' during LOAD

DEC	HEX	DESCRIPTION
43284	A914	Jump vector for SAVE
43382	A976	Jump vector for LOAD
43407	A98F	Get desired program
43639	AA77	Jump vector for EXEC
43682	AAA2	Beep
43723	AACD	Read 16 nulls + `1' from tape header
43768	AAF8	Long delay routine
43781	AB05	Write 63 nulls + `1' - tape header info
43841	AB41	2400 hertz out
43875	AB63	1200 hertz out
43980	ABCC	300 baud Cassette output vector
43987	ABD3	1200 baud Cassette output vector
44100	AC44	300 baud Cassette input vector
44104	AC48	1200 baud Cassette input vector
44204	ACAC	Reserved input vector
44215	ACB7	Jump vector for CURS
44255	ACDF	Jump vector for AUTO
44379	AC5B	Jump vector for HIRES
44385	AD61	Jump vector for SET
44401	AD71	Jump vector for RESET
44409	AD79	Jump vector for INVERT
44417	AD81	Jump vector for POINT
44550	AE06	Jump vector for USED
44952	AF98	Jump vector for LORES
45152	B060	Jump vector for PLOT
45595	B21B	Jump vector for RENUM
46256	B4B0	Jump vector for DELETE
46355	B513	Jump vector for GX
46584	B5F8	Parallel port output vector
46609	B611	Jump vector for EDASM
46617	B61C	Jump vector for MEM/NET
46647	B637	Jump vector for ERROR
47196	B85C	Power-up message: "Applied Technology" etc
47821	BACD	Start of initial set of values for 6545 registers; above values are copied to D2 to E1
47844	BAE4	Sound frequency data
48628	BDF4	Reset/Control sequences
48646	BE06	Jump to Monitor (if fitted)
48651	BE11	Self-test on BASIC 5.10
49151	BFFF	End of BASIC or CP/M Transient Program area
49152	C000	EDASM/Word-Bee if fitted
49155	C003	EDASM Monitor entry point
54129	D371	Word-Bee version 1.0 tape copy routine
54477	D4CD	Word-Bee version 1.2 tape copy routine
57205	DF75	EDASM tape copy routine
57344	E000	NETwork/Boot ROM if fitted
61439	EFFF	End of NETwork/Boot ROM
61440	F000	Start of screen RAM and Character gen Data
62463	F3FF	End of screen memory (64 x 16)
62464	F400	Start of 17th line (64 x 16)
62527	F43F	End of 17th line (64 x16)
63487	F7FF	End of screen RAM and Character gen Data
63488	F800	Start of PCG or Colour RAM
65535	FFFF	End of PCG or Colour RAM

■

BASIC Scratch Area

```
BFF5    :;-----------------------------------
BFF5    :;      BASIC SCRATCH AREA
BFF5    :;-----------------------------------
0000    :               org     scrtch
0000    :; 100H boundary ..
0000    :hash_table     equ     .
0000    :               defs    128
0080    :;Need {CTC} vectors to 10H boundary ..
0080    :move_here      equ     .       ;NZ scratch start
0080    :ctcv1  defs    4*2
0088    :piov1  defs    2
008A    :piov2  defs    2
008C    :break_disab    defs    1       ;disables break key if nz here
008D    :save_disab     defs    1       ;disables SAVE etc if NZ
008E    :               defs    '20'-(6*2)-2-7  ;reserved
0099    :col_flag       defs    1       ;=255 if colour BEE, else 0
009A    :col_mrsl       defs    1       ;color mode byte result
009B    :col_rslt       defs    1       ;normal byte result
009C    :col_fore       defs    1       ;foreground color
009D    :col_back       defs    1       ;background color
009E    :col_mode       defs    1       ;color mode
009F    :ucl_flag       defs    1       ;control UC mode of LIST
                                        ; nz=list in lower
00A0    :;
00A0    :stack  defs    2       ;top of memory = stack
00A2    :rst_jump       defs    2       ;warm-start jump address
00A4    :chk_byte       defs    2       ;55AA if initialized
00A6    :save_exec      defs    2       ;machine language exec address
00A8    :jp_mem defs    3       ;modifiable jump vectors for keywords
00AB    :jp_net defs    3
00AE    :jp_edasm       defs    3
00B1    :               defs    1       ;reserved
00B2    :;
00B2    :; this is the input and output vector tables ..
00B2    :out_tab        defs    2*8
00C2    :in_tab defs    2*8
00D2    :;
00D2    :; this is the crtc table, copied to ram for esc a,s,w,z
00D2    :crtc_tab       defs    2
00D4    :crtc_hor       defs    5
00D9    :crtc_ver       defs    3
00DC    :crtc_curs      defs    6       ;cursor control byte
```

For all you Assembly Language programmers, these BASIC scratch area locations should make life a little easier. Thanks again to Applied Technology, we've been able to reprint them from the 'Microbee Technical Manual'.

```
00E2    :;
00E2    :out_dev        defs    1       ;selects output from above table
00E3    :outl_dev       defs    1       ;select lprint output
                                         from out_tab
00E4    :in_dev defs    1       ;selects input from in_tab
00E5    :;
00E5    :vdmode defs    1       ;video mode control byte
00E6    :speed  defs    1       ;vdu delay period
00E7    :chars_used     defs    1       ;my addition for USED, etc.
00E8    :plot_type      defs    1       ;has "R","I" or space
00E9    :lo_cycle       defs    1       ;controls tape routine speed
00EA    :RS_baud        defs    1       ;controls RS232 speed
00EB    :; this is a shared scratch : graphics plot / tape routines
00EB    :gr_pseudo      equ     .
00EB    :required       defs    6       ;the load name to match
00F1    :header defs    6       ;dgos type header block
00F7    :fltype defs    1       ;file type character
00F8    :flleng defs    2       ;length of file
00FA    :flstrt defs    2       ;normal starting address
00FC    :flauto defs    2       ;jp to this address if flexec nz
00FE    :flspeed        defs    1       ;0 -> 300bd, nz -> 1200 bd
00FF    :flexec defs    1       ;is nz if this is auto execute
0100    :flprotect      defs    1
0101    :;
0101    :; From here on the scratchpad may change between versions
0101    :alpha_rev      defs    1       ;non z -> alpha reversed
0102    :key_down       defs    1       ;this key-code is down
0103    :rept_count     defs    2       ;controls speed of repeat key
0105    :rept_flip      defs    1       ;defines if retrace or not
                                         at last look
0106    :last_ascii     defs    1       ;keeps last char send
                                         by scn_in for repeat
0107    :keep_char      defs    1       ;keeps an ASCII code
                                         for inkey$'s use
0108    :kbds           defs    1       ;for port a keyboard
0109    :port_ack       defs    1       ;zero if parallel port out
                                         usable
010A    :buff_count     defs    1       ;no. bytes in cassette
                                         redirect buffer
010B    :cursor defs    2       ;cursor address 'F000'+
010D    :esc_flag       defs    1       ;nz if last vdu_out char
                                         was escape
```

BASIC Scratch Area

```
010E    :holdde defs    2       ;save de during ops
0110    :showcr defs    1       ;let list-line do inverse
0111    :trwd           defs    1       ;ignore o/p ovf STR
                                         if -1 else max/line
0112    :svelps defs    2
0114    :sveaut defs    2       ;auto-storage
0116    :uplink defs    2       ;scratch for renum
0118    :sram           defs    2
011A    :rnd            defs    '10'    ;random number seed
012A    :load_opt       defs    1       ;holds load options
012B    :;
012B    :free           defs    1       ;pcg storage ..
012C    :vdu_addr       defs    2
012E    :new_char       defs    pcg_charz
013E    :contpos        defs    2       ;continue address (de)
0140    :contstart      defs    2       ;start of currently
0142    :err_trap       defs    2       ;=line num if ON ERROR active
0144    :err_code       defs    1
0145    :err_line       defs    2       ;status of last error
0147    :;
0200    :               org     (.>8+1)<8
0200    :; 100H boundary ..
0200    :;*** from here may be overloaded by machine language progs. ***
0200    :;* this should be at scratch+'200'
0200    :ref_counts     defs    n_pcg_chars*2
0300    :free_recs      defs    n_pcg_chars*2
0400    :;
0400    :; Need 100H boundary           org     (.>8+1)<8
0400    :astck  defs    '100'
0500    :avar           defs    '1B0'
06B0    :ivar           defs    26*2
06E4    :fnstor defs    '10'
0700    :               org     (.>8+1)<8
0700    :;      Force to 100H boundary
0700    :fpbuff defs    '28'
0728    :ibufs  defs    '100'-'28'
0800    :obufs  defs    'C0'
08C0    :
08C0    :sd             defs    1
08C1    :fw             defs    1
08C2    :dp             defs    1
08C3    :tmpl           defs    2
```

```
08C5    :tmp2           defs    2
08C7    :count  defs    1
08C8    :               defs    1               ;### M.S.
08C9    :auto           defs    1
08CA    :crlbl  defs    2
08CC    :tmp3           defs    2
08CE    :tmp4           defs    2
08D0    :pbgn           defs    2
08D2    :pend           defs    2
08D4    :stlvl  defs    2
08D6    :optr           defs    2
08D8    :vstrt  defs    2
08DA    :tmpa           defs    2
08DC    :sstrt  defs    2
08DE    :xswe           defs    1
08DF    :xswl           defs    1
08E0    :prmt           defs    1
08E1    :sprmt  defs    1
08E2    :tmpc           defs    1
08E3    :zone           defs    1
08E4    :mode           defs    1
08E5    :tmpd           defs    2
08E7    :tmp5           defs    2
08E9    :dloc           defs    2
08EB    :flags  defs    1
08EC    :lims           defs    2
08EE    :autstp defs    1
08EF    :autln  defs    2
08F1    :tmp8           defs    2
08F3    :tmp9           defs    2
08F5    :ptrps  defs    2
08F7    :;stack moved (shouldn't be in table C), replaced by ..
08F7    :autdef defs    2       ;default for edit
08F9    :tmpf           defs    2
08FB    :sblvl  defs    1
08FC    :odvce  defs    1
08FD    :contln defs    2       ;CONT line No
08FF    :               defs    1       ;space filler
0900    :pstrt  equ     .
0900    :
0900    :end            equ     .
0900    :               end
```

■

User Groups

NEW SOUTH WALES

Sydney Microbee User's Group
Post Office Box C233,
Clarence Street,
Sydney 2000.
Contact: Mr Colin Tringham (President/Editor) on (02) 92 6408.

Primary Schools Microbee User's Group
C/- Denistone East Primary School,
Denistone East 2212.
Contact: Mr Barry Manefield (Co-ordinator).

Central Coast Computer Club
Koala Crescent,
West Gosford 2250.
Contact: Mr Max Maughan (Secretary).

Barrier Microbee User's Group
553 Radium Street,
Broken Hill 2880.
Contact: Mr P. R. Cotter (President) on (080) 88 1621.

Blue Mountains Microbee User's Group
C/- 28 Frank Street,
Mt Druitt 2770.
Contact: Mr Don Cunningham (Secretary).

Dubbo Microbee User's Group
C/- 33-35 Carrington Avenue,
Dubbo 2830.
Contact: Mr Frank Wilcox (Publicity Officer) on (068) 82 2722.

Newcastle Microbee User's Group
C/- 12 Cleverton Close,
Warners Bay 2282.
Contact: Lee Osman (Organiser) on (049) 48 8813.

Illawarra Microbee User's Group
C/- 49 Beatus Street,
Unanderra 2526.
Contact: Mr Ronald Read (Organiser) on (042) 71 2384.

Summerland Computer Club
C/- 112 Casino Street,
South Lismore 2480.
Contact: Diana Estreich (Co-ordinator).

Hawkesbury Microbee Computer Club
C/- 6 Warks Road,
Kurrajong Heights 2758.
Contact: Mr Bruce Rennie (Organiser) on (045) 67 7329.

Albury/Wodonga Microbee User's Group
202 Kooba Street,
Albury 2640.
Contact: Mr E. Eulenstein (Convenor) on (060) 25 1601.

Cooma Microbee User's Group
C/- Box 92,
Cooma 2630.
Contact: Mr Phill Zikan (Convenor) on (0648) 2 3315.

Wagga Wagga Microbee User's Group
C/- 1 Ilex Street,
Wagga Wagga 2650.
Contact: Mr Arthur Hand (Organiser) on (069) 22 4640.

Yass Microbee User's Group
25 DeMestre Street,
Yass 2582.
Contact: N. Burke (Hon Secretary).

QUEENSLAND

Gold Coast Microbee User's Group
1/100 Imperial Parade,
Labrador 4215.
Contact: Mr Col McLaren (Spokesperson) on (075) 31 4610.

Brisbane Microbee User's Group
PO Box 332,
Ashgrove 4060.
Contact: John Fisher (Secretary) on (075) 209 9830.

Cairns District Microbee User's Group
21 Marr Street,
Edmonton 4869.
Contact: Chas Eustance (Secretary) on (070) 55 4531.

Maryborough Microbee User's Group
218 Adelaide Street,
Maryborough 4650.
Contact: Mr Tony McCrae (Spokesperson) on (071) 22 1743.

Townsville Microbee User's Group
C/- Town and Country Computers,
CTL Centre,
Anne Street,
Aitkenvale 4814.
Contact: J. F. Johnson (Spokesperson).

Mackay Microbee User's Group
PO Box 230,
Mackay 4740.
Contact: Mr Geoff Gehrins (Spokesperson) on (079) 42 3214.

ISHS Computer Club
C/- Inala State High School,
Cnr Hampton Street and Glenala Road,
Inala 4077.
Contact: Glen Beaumont (President).

You probably can do it all on your own ... but you'll get there quicker if you join a user group. 'ETI' and 'Your Computer' columnists are always raving about the benefits of joining a user group, and as far as we can see they're right. As a user group member you'll meet other people who've been stung by the Bee, you'll have a pool of knowledge to draw on when you strike problems (and probably a pool of spare parts, too), and user groups often have access to technical and other information that lone hackers would have to find out for themselves. Here are the people to contact.

ACT

Canberra Microbee User's Group
C/- Microbee Computer Centre,
Upper Level,
Cooleman Court,
Weston 2611.
Contact: Mr Bill Horsfall (Convenor)
(062) 58 3193.

TASMANIA

Hobart Microbee User's Group
C/- 18 Tunah Street,
Howrah 7018.
Contact: Mr Brian Links (Assistant Secretary).

NORTHERN TERRITORY

Alice Springs Microbee User's Group
Post Office Box 3230,
Alice Springs 5750.
Contact: Mr Doug Craigie (Spokesperson).

Darwin Microbee User's Group
C/- 35 McCusker Avenue,
Enfield 5085.
Contact: Mr R. J. Cockburn (Executive Officer).

WESTERN AUSTRALIA

Microbee's User's Group of Western Australia
C/- 4 Garnkirk Road,
Greenwood 6024.
Contact: Mr Mike Oborn (Secretary) on
(09) 326 1261.

VICTORIA

Microbug Australia
PO Box 157,
Nunawading 3131.
Contact: Mr Dixon Kenney on
(03) 873 4476.

Bendigo Microbee User's Group
38 Nolan Street,
Bendigo 3550.
Contact: Mr Bruce Wilson (Spokesperson).

Puckapunyal/Seymour Microbee User's Group (P/SMUG)
C/- Monash Drive,
Seymour 3660.
Contact: Mr G. R. Chinner (Spokesperson) on (057) 92 3165.

Western Suburbs Microbee User's Group
PO Box 88,
St Albans 3021.
Contact: Sue Ferrito (Secretary) on
(03) 367 6569.

SOUTH AUSTRALIA

Microbee User's Group of South Australia
GPO Box 767,
Adelaide 5001.
Contact: Mr R. Jackson (Secretary).

The Port Lincoln Microbee User's Group
C/- Lincomputer,
PO Box 345,
Port Lincoln 5606.

INTERNATIONAL

"Microbiten"
C/- Olle Ljungquist,
Bjornkarrsv. 72,
S-183 41,
Taby, Sweden.

Suppliers

It's all no good if you can't buy the machines, the peripherals, the software and the parts which allow you to indulge in your 'hacking' activities. The following is a list of Microbee Computer Shops and Microbee dealers. *(See also Shoparound on page 116).*

Microbee Computer Shops

- Koala Crescent,
  West Gosford 2250.
  Phone: (043) 24 2711.

- 1a Pattison Avenue,
  Waitara 2077.
  Phone: (02) 487 2711.

- 729 Glenferrie Road,
  Hawthorn 3122.
  Phone: (03) 819 5288.

- 141 Stirling Highway,
  Nedlands 6009.
  Phone: (09) 386 8240.

- 151 Unley Road,
  Unley 5061.
  Phone: (08) 272-1384.

- 455 Logan Road,
  Stones Corner 4120.
  Phone: (07) 394 3688.

Microbee Dealers

Australbee
265a Springvale Road,
Glen Waverley 3150.

Central Data
14a Goodwin Street,
Invermay 7250.

Comput Ed
8 Park Arcade,
Park Avenue,
Coffs Harbour 2450.

Compu K
7 Casino Street,
Lismore 2480.

Electrographic Office Supplies
110 Abbott Street,
Cairns 4870.

Jaycar
117 York Street,
Sydney 2000.

Jaycar
115 Parramatta Road,
Concord 2137.

Jaycar
121 Forest Road,
Hurstville 2220.

Jaycar
Cnr Carlingford and Pennant Hills Roads,
Carlingford 2118.

Key Computer
77 Grenfell Street,
Adelaide 5000.

Key Computer
1061 South Road,
Edwardstown 5039.

Lincoln Computer Centre
25 Washington Street,
Port Lincoln 5606.

Pine Street Trading
54 Pine Street,
Mt Tom Price 6751.

Software 80
105 Milton Road,
Milton 4064.

Timberton Computers
94 High Street,
Wauchope 2446.

Town and Country Computers
CTL Centre,
Anne Street,
Aitkenvale,
Townsville 4814.

EDUCATION PRIORITY

microbee® classroom

networks for learning

Learning in a **microbee classroom** is fast, fun and effective. Students enjoy learning and advance rapidly.

The new **microbee electronic classroom network systems** are engineered to meet the demanding specifications set by Australian Education Departments. They are designed, developed and manufactured in Australia to rigid quality standards. Every **microbee** can be used either individually or in classroom networks. Schools can buy two or three **microbee 16K educators** or **32k communicators** now and upgrade to networking at any future time. All **microbees** are supplied with built-in BASIC, WORDBEE and the new TELCOM software that enables a **microbee** to 'talk' to another **microbee.** You network within classrooms or across the world!

Australian Education Software Support
There is a wide and ever growing range of high quality software written especially for Australian schools available for the **microbee.**
Just released is 'Birds of Antartica' — an outstanding database application for secondary students. A full list of all available teacher support resources for **microbee** is available from your local **microbee Computer Centre.**

SHIPWRECK ISLAND An intriguing adventure situation for 1 to 5 students Requires co-operative decision making

LEMONADE STALL A 'business' situation Calculated and agreed upon decisions will achieve success in various weather conditions

T.N.T. Highly motivating for all ages Chosen tables at selected speeds in a race against an explosive fuse

GEOGRABEE All students will enjoy trying to beat the clock whilst identifying oceans, continents and countries

microbee APPLIED TECHNOLOGY RETAIL PTY LTD

DIRECT ORDERS
PHONE (02) 487 2711
TELEX AA72767

microbee Computer Centres Australiawide

Everything a micro should bee